

# Smidig IT-2

**Programmering og systemutvikling med Python**



**i Visual Studio Code**



Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Forord

Hei, og velkommen til Smidig IT-2! Dette digitale læreverket er spesielt utviklet for å imøtekomme læreplanen i Informasjonsteknologi 2 (LK20<sup>1</sup>). Målet er å tilby et smidig og skreddersydd undervisningsopplegg som sikrer en bred forståelse av faget, inkludert samfunnsmessige konsekvenser og etiske dilemmaer.

Framdriften i dette digitale læreverket er basert på prinsippene for [smidig systemutvikling](#). Vi går gjennom flere runder hvor vi jobber med alle stegene litt etter litt i hver runde, i stedet for å fullføre hvert steg før vi går videre til neste. I en tradisjonell lærebok ville du vanligvis ha fullført halvparten av kompetansemålene når du er halvveis i boken. På dette tidspunktet ville du ennå ikke ha lært noe om de andre kompetansemålene. Med vår smidige tilnærming jobber vi kontinuerlig med og lærer om alle kompetansemålene gjennom fire runder i løpet av skoleåret. Med andre ord, når du er midtveis, vil du ha fullført "halvparten av alt i stedet for alt av halvparten". Undervisningen blir mer variert, og det blir lettere å huske og forstå sammenhenger mellom ulike emner i faget.

Konfucius er kjent for blant annet sitatet: "Fortell meg, og jeg vil glemme. Vis meg, og jeg vil huske. La meg gjøre, og jeg vil forstå!". Disse visdomsordene inspirerer til en praktisk tilnærming. I dette læreverket vil du finne praktiske eksempler og oppgaver som gir deg muligheten til å reflektere, utforske og anvende din faglige kunnskap, også på en måte som fremmer dybdelæring.

Læreplanen for IT-2 legger vekt på en bred forståelse av faget, inkludert samfunnsmessige konsekvenser og etiske dilemmaer. Kompetansemålene omfatter objektorientert programutvikling, feilsøking og testing, samarbeid, standarder og vurdering av brukervennlighet. Objektorientert modellering, gjenbruk av programkode og vurdering av alternative løsninger er også viktige elementer. Du skal dessuten kunne programmere innhenting, analyse og presentasjon av informasjon fra reelle datasett.

Læreplanen representerer et læringsløft, og i LK20 er programmering integrert i andre fag som matematikk og realfag. Derfor prioriterer vi en rask gjennomgang av grunnleggende programmering for å gi tilstrekkelig tid til å arbeide med de nye kompetansemålene.

Oppsummeringene etter hver runde bekrefter hvordan innholdet dekker kompetansemålene. Samtidig styrkes grunnleggende ferdigheter gjennom øvingsoppgaver og praktisk anvendelse av reelle datasett og programmering. Det tverrfaglige temaet, demokrati og medborgerskap, blir spesielt adressert i ulike bolker gjennom drøfting av informasjonsteknologiens innvirkning på samfunnet, etiske dilemmaer og universell utforming.

Valget av Python som programmeringsspråk og Visual Studio Code (VS Code) som utviklingsplattform gir en solid plattform for å utforske og lære programmering, og er dermed godt egnet til å oppnå kompetansemålene i læreplanen. Python er et anerkjent, enkelt og svært populært språk, og kunnskap om Python kan være verdifulle både i videre studier og i fremtidig karriere. VS Code, med sine mange utvidelser, gir et effektivt og

---

<sup>1</sup> Læreplanverket for Kunnskapsløftet 2020

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

fleksibelt arbeidsmiljø, og kan også brukes sammen med andre programmeringsspråk.  
Se [Python in Visual Studio Code](#).

I denne tredje utgaven har vi fjernet, endret og lagt til nytt innhold for å tilpasse læreverket til utviklingen i IT-2-faget. Enhetstesting er skrevet om og heter nå *Enhetstesting av objektorienterte programmer* (2.7). Det har blitt lagt til to nye bolker: den ene om *Reelle datasett med OOP og GUI* (3.9) og den andre om *Simuleringer* (4.4). *Geodata* (5.10) er tatt ut av Runde 4 og flyttet til Del 5. Innhold som ikke ble plass til, men som kan være nyttig til oppslag eller fordympning, er også samlet her. I denne delen har det kommet til noen mindre emner som *Tilfeldige tall og utvalg* (5.2), *Etterligning (Mocking) med pytest* (5.6) og *Unittest* (5.7). For de som ønsker å fordype seg mer, er det også viet mer plass til *Designmønstre* (5.8) og *API for web og mobil med Flask og Dash* (5.11).

Innholdsfortegnelsen inneholder nå bare hovedoverskriftene. Hver bok har sin egen innholdsfortegnelse og viser kompetansemålene som blir vektlagt. På slutten av hver bok er det lagt til et sammendrag kalt *Ta med minst dette*, som gir en kort oppsummering av det viktigste innholdet. Enkelte bolker har byttet plass i Runde 4, som også bør tilpasses framdriften i faget, avhengig av tilgjengelig tid. Utover dette er det rettet opp feil og gjort andre forbedringer.

Mysen, august 2024

Ola Lie

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## Innhold

Forord .....	2
Innhold .....	4
Forberedelser .....	7
Runde 1 .....	9
1.1 Grunnleggende programering.....	9
1.2 Datatyper, objekter, tekst og tall .....	13
1.3 Tekstformatering.....	17
1.4 Løkker.....	21
1.5 Lister .....	26
1.6 Filer.....	34
1.7 Feil .....	39
1.8 Modellering.....	43
1.9 Reelle datasett med CSV.....	48
1.10 Informasjonsteknologi.....	55
1.11 Oppsummering .....	60
Runde 2 .....	62
2.1 Grafisk brukergrensesnitt med tkinter.....	62
2.2 Tupler og ordbøker .....	72
2.3 Versjonskontroll med Git .....	81
2.4 Funksjoner .....	92
2.5 Programvaretesting .....	103
2.6 Objektorientert programering .....	111
2.7 Noen enkle verktøy for systemutvikling .....	121
2.8 Utveksling av data med JSON.....	128
2.9 Reelle datasett med JSON .....	135
2.10 Deepfakes.....	139
2.11 Oppsummering .....	144
Runde 3 .....	147
3.1 Jupyter, NumPy og pandas .....	147
3.2 Diagrammer med seaborn .....	154
3.3 Sett og boolsk algebra.....	163
3.4 Tekst, grafikk, animasjon og lyd med Pygame .....	171

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

3.5 Klassediagram.....	181
3.6 Dokumentasjon .....	192
3.7 Enhetsesting av objektorienterte programmer.....	199
3.8 Fortsettelse av Pandas .....	211
3.9 Reelle datasett med OOP og GUI.....	222
3.10 Kunstig intelligens.....	230
3.11 Oppsummering .....	237
Runde 4 .....	240
4.1 Brukervennlighet.....	240
4.2 Hendelser , kollisjoner, tidsstyring og GUI-integrering i Pygame.....	246
4.3 Datarensing og datautforsking.....	255
4.4 Datasimuleringer.....	267
4.5 Standarder for datahåndtering.....	276
4.6 Flere verktøy for samarbeid.....	285
4.7 Mer om OOM.....	293
4.8 Mer om testing .....	300
4.9 Reelle datasett med REST API.....	308
4.10 Posisjonsdata .....	316
4.11 Oppsummering .....	321
Del 5.....	323
5.1 Dato og tid .....	323
5.2 Tilfeldige tall og utvalg.....	327
5.3 Sortering.....	331
5.4 Rekursjon .....	334
5.5 Avansert JSON .....	336
5.6 Etterligning (mocking) med pytest .....	339
5.7 Unittest .....	341
5.8 Designmønste .....	343
5.9 Hendelser, animasjon, kollisjoner og simulering med tkinter .....	348
5.10 Geodata.....	356
5.11 API for web og mobil med Flask og Dash .....	365
Vedlegg 6 .....	375
6.1 Verktøy og metoder for systemutvikling .....	375

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

6.2 Før eksamen.....	376
6.3 Forankring i læreplanen .....	379
6.4 Kompetanse mål.....	382

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Forberedelser

Før vi går i gang, må vi sørge for at vi har installert de nødvendige programmene.

#### Installer den nyeste versjonen av [Python](#)

1. **Last ned og installer Python:**

<https://www.python.org/downloads/>

Huk av for å legge python til miljøvariabelen PATH

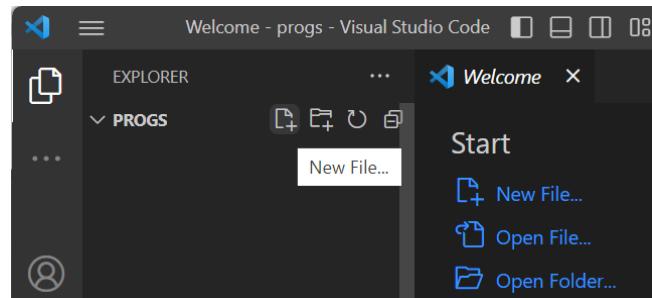
2. **Sjekk installasjonen:** Åpne terminalvinduet og skriv `python -V`, som skal vise versjonsnummeret på skjermen.

```
Ledetekst
C:\Users\olali>python -V
Python 3.10.6
```

#### Installer [Visual Studio Code for Windows](#)

1. **Last ned og installer VS Code:** <https://code.visualstudio.com/download>

2. **Opprett en programmappe:** for programmer, f.eks. `IT-2\progs` i [Dokumenter](#), med filutforskeren eller samtidig som du åpner en mappe i VS Code med *File, Open Folder...* fra menylinjen eller ved å klikke på *Open Folder...* ikonet på velkomstsiden.



3. **Opprett en Python-fil:** Velg *File, New File...* fra menylinjen eller klikk på et av *New File...* ikonene. Gi filen navnet `hallo.py`.
4. **Skriv inn kode:** `print('Hallo!')` i editoren og lagre programmet med *File, Save* (Ctrl+S)
5. **Installer anbefalte utvidelser for Python:** Klikk *Install* nede til høyre i vinduet på spørsmål om å installere anbefalte utvidelser for Python.
6. **Kjør programmet:** Klikk på pilen opp til høyre eller velge *Run, Run Without Debugging* (Ctrl+F5) fra menylinjen. Dette vil åpne et panel med en "TERMINAL"-fane der utskriften fra programmet vises.

```
hallo.py - progs - Visual Studio C...
EXPLORER   PROGS
Get Started hallo.py
hallo.py
1 print('Hallo!')

JUPYTER: VARIABLES  PROBLEMS  OUTPUT  TERMINAL
Python + ×
PS C:\Users\olali\OneDrive - Viken fylkeskommune\dokumenter\_IT-2\progs>
> & C:/Users/olali/AppData/Local/Programs/Python/Python310/python.exe "c:/users/olali/onedrive - viken fylkeskommune/dokumenter/_IT-2/progs/hallo.py"
Hallo!
PS C:\Users\olali\OneDrive - Viken fylkeskommune\dokumenter\_IT-2\progs>
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Virtuelle miljøer

Når vi installerer Python og ulike biblioteker, blir de normalt lagt til i det samme globale miljøet på datamaskinen. Dette kan føre til utfordringer når vi oppdaterer enten Python eller bibliotekene med nyere versjoner. Noen ganger er disse oppdateringene ikke kompatible med hverandre eller med vår eksisterende kode, noe som kan resultere i at programmer som tidligere fungerte, plutselig krasjer.

For å takle denne problemstillingen, kan vi benytte oss av virtuelle miljøer. I et virtuelt miljø har vi full kontroll og kan selv bestemme (og låse) hvilke versjoner av Python og biblioteker vi vil bruke for hvert enkelt, spesifikke program. Dette gir oss muligheten til å unngå konflikter og sikrer at programmene våre forblir stabile, uavhengig av eventuelle globale oppdateringer.

Selv om bruk av virtuelle miljøer er en god og anbefalt praksis for Python prosjekter, har vi valgt å ikke inkludere denne tilnærmingen her. Vi skal utvikle mange mindre programmer som sjeldent tar mer enn et par timer å lage og som ikke er ment å være i bruk lengre enn gjennom skoleåret. I denne sammenhengen ville innføringen av virtuelle miljøer medføre unødvendig administrasjon og bruk av ressurser. Ved behov har vi alltid muligheten til å opprette virtuelle miljøer i etterkant, med de spesifikke versjonene vi ønsker å bruke. Informasjon om å opprette virutelle miljøer finnes blant annet på VS Code sine nettsider: <https://code.visualstudio.com/docs/python/environments>.

Vurderingsseksemplar

## Runde 1

### 1.1 Grunnleggende programmering.

#### Bolkens innhold

Innledning .....	9
Grunnleggende prinsipper i programmering.....	9
Utføring i Python .....	9
Variabler, tilordning og funksjoner.....	11
Ta med minst dette.....	11
Øvingsoppgaver.....	11

#### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

#### Innledning

Læreplanen i informasjonsteknologi 2 nevner ikke direkte basisferdigheter i programmering. Kompetansemålene er formulert på et høyere nivå, med vekt på å vurdere alternative løsninger, anvende relevante systemutviklingsmetoder, utvikle objektorienterte programmer, benytte strategier for feilsøking og testing, og mer. For å oppnå disse målene er det imidlertid nødvendig å ha en solid forståelse av de grunnleggende prinsippene innen programmering.

#### Grunnleggende prinsipper i programmering.

De fleste programmeringsspråk har mye felles:

- **Inndata:** tar imot data fra tastatur, mikrofon, kamera, leser inn data fra en fil, ...
- **Utdata:** viser data på skjermen, skriver ut data på papir eller til en fil, spiller lyd, ...
- **Beregninger:** matematiske operasjoner
- **Valgsetninger:** utfører spesifikke instruksjoner når visse betingelser er oppfylt
- **Løkker:** utfører samme instruksjoner gjentatte ganger med mindre variasjoner.

#### Utføring i Python

Vi har allerede programmert **utdata** med `print()`. La oss ta imot **inndata** fra tastaturet med `input()`. Først skriver vi inn et navn som lagres i **variabelen** `navn`. Deretter slår vi sammen tre tekststrenger med `+` tegnet og skriver det hele ut med `print()`. `print()` og `input()` er innebygd funksjoner, også kalt *built-in functions* eller BIF, som følger med Python. Her er en [liste over alle innebygde funksjoner](#).

```
1  navn = input('Hva heter du? ')
2  print('Hallo ' + navn + '!')
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Vi bruker standard matematiske operatører. **Beregningene** og variabelbruk er forklart i kommentarene som begynner med `#` tegnet.

```
1  a = 1 + 2  # 3 lagres i a
2  b = a + 3  # hent a (3), legg til 3
3  |           # og lagre resultatet (6) i b
4  a = 4       # lagre 4 i a
5  print(a)   # skriv ut a (4)
6  print(b)   # skriv ut b (6, ikke 7)
```

**PEMDAS** står for Parenteser, Eksponenter, Multiplikasjon, Divisjon, Addisjon og Subtraksjon: `( )`, `**`, `*`, `/`, `+` og `-`. Operasjonene utføres i denne rekkefølgen. Regn ut uttrykkene i tabellen til høyre uten hjelpeMidler. Sjekk deretter svarene i Python, for eksempel ved å skrive `python` i Terminal-vinduet for å starte fortolkeren. Avslutt fortolkeren med `Ctrl+Z` og `Enter`.

Det finnes flere operatorer enn de nevnt ovenfor: Se [Operators](#). Parenteser er ikke operatorer, men skilletegn (*delimiter*).

Vi vil ofte møte situasjoner der vi må skrive forskjellige kode avhengig av spesifikke betingelser (vilkår), som må undersøkes. For eksempel, hvis noen er under 16 år, kan de få barnebillett. Hvis noen handler for mer enn tusen kroner, kan de få gratis frakt, osv. Slike **valgsetninger** kodes med `if`-setninger i Python. Normal programflyt er setning for setning etter hverandre.

I valgsetningen til høyre utføres koden bare i én av to veier, før veiene møtes igjen. Endre alderen til 20 og se hva som skjer.

Husk at Python bruker innrykk for å gruppere kode i blokker. I andre programmeringsspråk brukes ofte krøllparentenser `{ }`. Vi kan tenke på setninger (kode, instruksjoner) i en blokk som én setning.

En datamaskin kan utføre milliarder av operasjoner i sekundet. Selv om vår hjerne fortsatt er flinkere enn datamaskinen, klarer vi ikke å regne like fort. Ofte er det samme kode som gjentas flere ganger, og det kan vi oppnå med **løkker** og ofte med lister eller tabeller. Koden til høyre viser en `for`-løkke i Python<sup>2</sup>.

```
1  alder = 10      # Denne setningen kjøres alltid
2
3  if (alder<16):
4      pris = 15    # Enten så kjøres denne setningen
5  else:
6      pris = 25    # Ellers så kjøres denne
7
8  print(pris)    # Denne setningen kjøres alltid
```

```
1  liste = [3,8,5,11,7]
2  sum = 0
3
4  for tall in liste:
5      sum = sum + tall
6
7  print(liste) # [3, 8, 5, 11, 7]
8  print(sum)  # 34
```

<sup>2</sup> Vi bør ikke bruke `sum` som et variabelnavn i Python, for det vil overskygge den innebygde funksjonen `sum()`. Bruk heller `sum_` eller bedre, et mer beskrivende variabelnavn.

Uttrykk	Resultat
<code>2+3*4</code>	
<code>2*3+4</code>	
<code>2*(3+4)</code>	
<code>4*3/2</code>	
<code>4/2*3</code>	
<code>1-2/3</code>	
<code>1/3-2</code>	
<code>1/(3-2)</code>	
<code>2**3-1</code>	
<code>2**2*(3-1)</code>	
<code>2*3**4</code>	
<code>2**3*4</code>	

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi må utføre samme setninger flere steder i programmet, kan det være lurt å legge disse kodelinjene inn i en **funksjon** av flere årsaker (hvilke?). La oss si at vi ønsker å finne det største tallet i en tabell. I Python må vi definere funksjoner før vi bruker dem. Vi kan tenke på en funksjon som en navngitt blokk med kode. Når funksjonen kalles, utføres denne koden. Funksjoner inngår i kompetansemålet "generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode". Senere vil det kun stå, se [KM07](#).

```
def finn_maks(liste):
    maks = liste[0]      # Det første
                           # tallt i lista
    for tall in liste:
        if tall > maks:
            maks = tall

    return maks

liste1 = [3,8,5,11,7]
print(liste1)
print(finn_maks(liste1)) # Skriver ut 11

liste2 = [1,13,4,5,3,3,8,5,11,7]
print(liste2)
print(finn_maks(liste2)) # Skriver ut 13
```

### Variabler, tilordning og funksjoner

Tenk på en **variabel** som merkelappen på en boks som kan inneholde data (lagringsplass i RAM). Dessuten betyr ikke **likhetstegnet**, "er lik", men "sett lik". Regn ut høyresiden av likhetstegnet først. Resultat lagres i en variabel som står til venstre for likhetstegnet. Innholdet til variabler til høyre for likhetstegnet inngår i beregningene.

Hvis vi gjentar samme kode flere steder i programmet, kan vi legge denne koden inn i en **funksjon**. Der hvor koden skulle stått, kaller vi i stedet opp funksjon. Når koden i funksjonen er utført, fortsetter programmet etter der vi kalte opp funksjonen.

### Ta med minst dette

Læreplanen forutsetter basisferdigheter i programmering. De fleste programmeringsspråk har mye felles: inndata, utdata, beregninger, valgsetninger og løkker. Det er også nødvendig å kjenne til variabler, tilordninger (=) og funksjoner, samt å vite at Python bruker innrykk i stedet for krøllparenteser for kodeblokker. Innledningsvis vil vi likevel ha med bolker med flere detaljer om hvordan disse grunnleggende prinsippene utføres i Python.

### Øvingsoppgaver

#### 1.1.1

Skriv et program som regner ut hvor mye vekslepenger kunden skal ha tilbake når

```
pris = 673
betalt = 1000
```

#### 1.1.2

Skriv et program som regner ut volumet av en sylinder i liter ( $\text{dm}^3$ ) når målene er oppgitt i cm

```
diameter = 10 # cm
høyde = 15 # cm
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.1.3

Utforsk programmet nedenfor ved å skrive inn ulike navn og kjøre programmet. Prøv også med store og små bokstaver, siffer og spesialtegn i navnene. Skriv ned hva du fant ut.

```
1  navn1 = input('Skriv inn det første navnet: ')
2  navn2 = input('Skriv inn det andre navnet: ')
3
4  if navn1 > navn2:
5      print(navn1 + ' er større enn ' + navn2)
6  else:
7      print(navn2 + ' er større eller lik ' + navn1)
```

### 1.1.4

Bruk eksemplene om beregninger og løkker i teksten til å legge sammen tallene fra 1 til 10 på to forskjellige måter.

Python har den innbygde funksjonen [range](#) som kan lage sekvenser av tall. Koden `range(1,11)` lager en sekvens av tallene fra og med 1 til og med 10. [range](#) kan brukes på samme måte som en liste i [for](#)-setningen. Prøv å regne ut den samme summen ved bruk av [range](#)-funksjonen.

Python har også den innbygde funksjonen [sum](#) som kan legge sammen elementene i sekvenser. Prøv å regne ut den samme summen ved bruk av [sum](#)-funksjonen.

Vurder de ulike metodene basert på [KM03](#). Hvilken metode foretrekker du? Hvordan påvirkes vurderingen dersom tallene går fra 1 til 10 000?

Kunnskap om rekker i matematikken gjør det mulig å løse denne oppgaven på nok en ny måte.

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.2 Datatyper, objekter, tekst og tall

#### Bolkens innhold

Innledende eksempel.....	13
Grunnleggende datatyper.....	13
IntelliSense i Visual Studio Code .....	13
Klasser, objekter, egenskaper og metoder .....	14
Ta med minst dette.....	15
Øvingsoppgaver.....	16

#### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv

#### Innledende eksempel

"Programmer en rutine som beregner flytid når brukeren skriver inn avstand og fart. (Fart oppgis i knop, og avstand oppgis i kilometer, 1 knop = 1,852 km/t.) Angi svaret i timer og minutter eller som et desimaltall" <sup>3</sup>.

Programmet til høyre vil generere følgende feilmelding: `TypeError: can't multiply sequence by non-int of type 'float'`. For å forstå denne feilmeldingen, må vi kjenne til ulike **datatyper** i programmering.

```
1  avstand_i_km = input('Oppgi avstand i km: ')
2  fart_i_knop  = input('Oppgi fart i knop: ')
3  fart_i_kmpt  = fart_i_knop * 1.852
4  flytid       = avstand_i_km / fart_i_kmpt
5  print(flytid)
```

#### Grunnleggende datatyper

Grunnleggende datatyper inkluderer heltall, desimaltall, tekst og boolske verdier (sann eller usann).

I Python kan vi bruke `type`-funksjonen ([BIF](#)) til å sjekke datatype til en verdi eller variabel. De tilsvarende datatypene i Python er `int`, `float`, `str` og `bool`.

```
>>> type(123)
<class 'int'>
>>> type(1.23)
<class 'float'>
>>> type('Askim')
<class 'str'>
>>> type(True)
<class 'bool'>
```

#### IntelliSense i Visual Studio Code

```
b_1_2_1 (variable) avstand_i_km: str
1  avstand_i_km = input('Oppgi avstand i km: ')
```

Hvis vi holder musepekeren over `avstand_i_km` i programmet, vises informasjonen som indikerer at variabelen er av typen `str`. Disse hjelpeinformasjonene (*code hints*) er en del av [IntelliSense-funksjonen](#) i Visual Studio Code.

<sup>3</sup> Fra IT-2 eksamen våren 2010 (LK06)

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

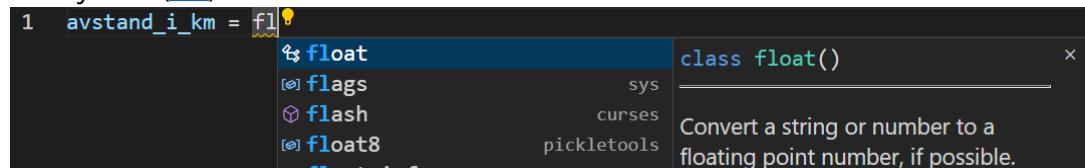
Hvis vi holder vi musepekeren over `input`-funksjonen, vises informasjonen som indikerer at den returnerer en tekst (`str`).

```
input('Oppgi avstand i km: ')
(function) input: (__prompt: object = ..., /) -> str
Read a string from standard input. The trailing newline is stripped.
The prompt string, if given, is printed to standard output without a trailing newline before reading input.
If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError. On *nix systems, readline is used
if available.
```

Dette er årsaken til feilmeldingen: `input` returnerer en tekst, og det er ikke tillatt å multiplisere en tekst (sekvens av tegn) med et desimaltall. Hvis det hadde vært et heltall, ville det vært i orden.

```
>>> 3.0 * 'Hurra! '
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
>>> 3 * 'Hurra! '
'Hurra! Hurra! Hurra! '
```

Hvis teksten representerer et tall, kan vi konvertere den til et desimaltall ved å bruke `float`-funksjonen ([BIF](#)).



A screenshot of a Python code editor showing the `float` function being completed. The code line is `1 avstand_i_km = float(`. A tooltip appears with the documentation: "Convert a string or number to a floating point number, if possible." Below the tooltip, the `float` class definition is shown: `class float():`

Når vi skriver, gir [IntelliSense](#) oss forslag for automatisk fullføring, også kjent som *auto complete*. Vi kan velge alternativene med **piltastene** og fullføre med **Tab** eller **Enter**, eller vi kan fortsette å skrive selv. Vi kan også få tilgang til [IntelliSense](#) ved å bruke **Ctrl+mellomrom**-kobminajonen.

Hvis vi nå gjør endringer i de to første linjene i programmet, vil det fungere bedre.

```
1 avstand_i_km = float(input('Oppgi avstand i km: '))
2 fart_i_knop = float(input('Oppgi fart i knop: '))
```

```
Oppgi avstand i km: 500
Oppgi fart i knop: 200
1.3498920086393087
```

### Klasser, objekter, egenskaper og metoder

I dokumentasjonen for `float` ser vi også at det står `class float()`. **klass** eller klasse er et sentralt begrep innen objektorientert programmering. En **klasse** er en mal (oppskrift) på hvordan vi lager et **objekt** av denne klassen. Tenk på en klasse som en avansert datatype som kan inneholde flere **variabler** og **funksoner**. Variablene kalles for objektets **egenskaper**

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

(*attributes*)<sup>4</sup> og funksjonene kalles for objektets **metoder**. Vi får tak i dem med **punktnotasjon** (*dot notation*). Se [KM05](#).

```
1 class Person:      # Opprett en klasse
2     fornavn = 'N'   # Attributt/egenskap
3     etternavn = 'N'
4
5 person = Person()  # Opprett et objekt
6 person.etternavn = 'Nilsen'           # Punktnotasjon
7 print(person.fornavn, person.etternavn) # N Nilsen
```

Alt i Python er et objekt, og hvert objekt er en instans (forekomst) av minst én klasse. Så fra nå av kan vi slutte å tenke på en variabel som merkelappen på en boks som kan inneholde data og heller innse at en **variabel er en referanse til et objekt**.

La oss gå tilbake til det aller første eksempelet. Vi vet nå at 'Hallo!' er et tekstobjekt som kan ha metoder som vi kan bruke med punktnotasjon. Når vi skriver et punktum etter 'Hallo!', viser [IntelliSense](#) egenskaper og metoder som kan benyttes for objekter av klassen `str`. Uten IntelliSense og kjennskap til objektets metoder, ville alternativet vært å lete i [str-dokumentasjonen](#). Vi ser at det finnes en metode `upper` som gjør teksten om til store bokstaver.

```
1 print('Hallo!')
```

title	upper() -> LiteralString
translate	upper() -> str
<b>upper</b>	
zfill	
__add__	
__annotations__	

Return a copy of the string converted to uppercase.

Husk å avslutte med metoden med `()`.

```
1 print('Hallo!'.upper()) # HALLO!
```

Et par andre eksempler kan være å telle opp antall 'm'-er i 'Smidig IT-2 med Python'

```
1 tittel = 'Smidig IT-2 med Python'
2 print(tittel.count('m')) # 2
```

og en sentrert tekst.

```
1 tittel = ' Smidig IT-2 med Python '
2
3 #----- Smidig IT-2 med Python -----
4 print(tittel.center(80, '-')) # 2
```

### Ta med minst dette

De grunnleggende datatypene i Python er `int` (heltall), `float` (desimaltall), `str` (tekst) og `bool` (boolske verdier). Hvis vi ikke bruker dem riktig, kan det oppstå feil i programmet. *IntelliSense* i *Visual Studio Code* er et nyttig verktøy som gir detaljert informasjon og forslag, både når vi fører musepekeren over ulike deler av koden, og når vi skriver, for automatisk fullføring. Klasser og objekter med egenskaper (variable) og metoder (funksjoner) er grunnleggende i objektorientert programmering. Alt i Python er objekter, og vi kan utføre objektenes metoder med punktnotasjon. Eksempelvis vil "`Harald.upper()`" bli til "`HARALD`". Senere skal vi lære å lage våre egne klasser.

<sup>4</sup> Her er egenskapene `fornavn` og `etternavn` tilknyttet selve klassen `Person`, og ikke et spesifikt objekt av klassen. Dette konseptet blir grundigere forklart i bolken [2.6 Objektorientert programmering](#).

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Øvingsoppgaver

I oppgavene til denne bolken møter vi også heltalsdivisjon (`//`) og modulo (`%`) operatorene. Heltalsdivisjon gir oss hvor mange hele ganger et tall går opp i et annet. Modulo gir resten i en heltalsdivisjon.

`13 // 5` gir `2`, og `13 % 5` gir `3` fordi  $13 = 2 \cdot 5 + 3$ .

#### 1.2.1

Bruk eksempelet om flytid og regn ut tiden i timer, minutter og sekunder. I denne oppgaven skal du kun bruke `*`, `/`, `+` og `-` samt `int()`.

Gjør om flytiden til sekunder og bruk `//` (heltalsdivisjon) og `%` modulo for å gjøre det samme.

Bruk [divmod](#) (BIF) til å gjøre det samme. `divmod` utfører både heltalsdivisjon og modulo.

Eksempel: `kvotient, rest = divmod(13,5)` gir `kvotient=2` og `rest=3`<sup>5</sup>.

Bruk [datetime.timedelta](#) klassen til å gjøre det samme. Den tillater at du oppgir antall timer som desimaltall og når vi gjør om objektet om til tekst med `str()`, får vi teksten på formatet `tt:mm:ss`. Sekundene vises med desimaler som kan fjernes. Husk å ha med `import datetime`.

#### 1.2.2

Lag en variabel som refererer til teksten 'Smidig IT-2 med Python'. Bruk IntelliSense til å utforske metodene til tekstobjektet. Beskriv spesifikt forskjellen mellom metodene `capitalize()`, `title()` og `lower()`. Bruk `print()` til å vise resultatene.

---

<sup>5</sup> Se 2.2 [Tupler](#) og ordbøker.

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.3 Tekstformatering

#### Bolkens innhold

Ulike måter å formtere tekst.....	17
Mer om f-strings.....	17
Ta med minst dette.....	19
Øvingsoppgaver.....	19

#### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

#### Ulike måter å formtere tekst

Vi har sett at vi kan skjøte tekstobjekter med `+` operatoren:

```
'Hallo ' + navn + ' !'
```

Tekst kan formateres på flere måter. Selv om vi bruker bare en metode, vil det være nyttig å kjenne til flere metoder for å forstå andres kode. Formatering inngår dessuten i kompetansemålet om å "utforske og vurdere alternative løsninger for design...", se [KM03](#).

Skal vi skjøte inn tall i tekststrenger, må tallene først gjøres om til tekst med [str \(BIF\)](#), eventuelt adskille tekst og tall med komma.

```
1 navn = 'Per'
2 alder = 17
3
4 # Jaså Per, du er 17 år.
5 print('Jaså ' + navn + ', du er ' + str(alder) + ' år.')
```

Tidligere brukte man `%` operatoren:

```
print('Jaså %s, du er %s år.' % (navn, alder))
```

Senere kom `str.format` som er mer regelmessig:

```
print('Jaså {}, du er {} år.'.format(navn, alder))
```

Vi skal bruke **f-strings**:

```
print(f'Jaså {navn}, du er {alder} år.')
```

#### Mer om f-strings

Bruken av formatert tekst gjør programmet også sårbart, for det muliggjør tilgang til vilkårlige variabler i programmet<sup>6</sup>. Dette kan unngås med [Template strings](#) som ikke tillater formatspesifikasjoner.

<sup>6</sup> Se [Python format string vulnerabilities](#)

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Det er et hav av muligheter når vi skal formattere tekst. Hovedsyntaksen ser slik ut:

```
"{[ field_name ]|[!" conversion] [": format_spec] }"
```

[!" conversion] blir sjeldent brukt. [": format\_spec] er mer vanlig:

```
[[fill][align][sign][#[O][width][grouping_option][.precision][type]
```

Detaljene finner vi i [Format Specification Mini-Language](#):

```
format_spec ::= [[fill][align][sign]["z"]["#"]["0"]][width][grouping_option][. precision][type]
fill       ::= <any character>
align      ::= "<" | ">" | "=" | "^"
sign       ::= "+" | "-" | ""
width      ::= digit+
grouping_option ::= "_" | ","
precision  ::= digit+
type       ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"
```

For heltall er eksempelvis type **b** binært, **d** talltalsystemet og **x** hex.

```
tall = 13

# Tallet 13 er 00001101 binært.
print(f'Tallet {tall} er {tall:08b} binært.') # fill=0, width=8 og type=b

# Tallet 13 er 0d heksadesimalt.
print(f'Tallet {tall} er {tall:02x} heksadesimalt.') # fill=0, width=2 og type=x
```

Vi kan velge høyrejustering og antall desimaler med mer.

```
verdier = [1.1, 22.22, 333.333]

#    1.10
#   22.22
#  333.33
for tall in verdier:
    print(f'{tall:>10.2f}') # align=>, width=10, .precision=2 og type=f
```

tall = 1/3  
print(round(tall,2)) # 0.33

Alternativt kan vi runde av tallet med [round \(BIF\)](#). [round](#) runder ned og opp.

[Int \(BIF\)](#) fjerner desimalene. Utforsk også [math.ceil](#) og [math.floor](#).

```
>>> round(2.3)
2
>>> round(2.7)
3
>>> int(2.3)
2
>>> int(2.7)
2
```

Valget mellom **f-strings** og **.format** avhenger stort sett av om vi prioritører enkelhet og ytelse (f-strings) eller støtte for tidligere versjoner og muligheten for gjenbruk av variabler (.format). f-strings er enklere fordi variablene og formateringen er samlet i samme streng. Dermed slipper vi å hoppe mellom strengen og .format for å sjekke hva som hører til hvor, spesielt når det blir flere variabler. f-strings utføres vanligvis raskere, men i praksis utgjør dette liten forskjell. Undersøk selv hvor lang tid det tar å kjøre koden vist i bildet over til høyre (b\_1\_3\_3\_timit.py). Å forstå .format

```
import timeit

def txt_format():
    return 'Hei, jeg er {} og {} år.'.format('Oda', 23)

def txt_fstring():
    navn, alder = 'Oda', 23
    return f'Hei, jeg er {navn} og {alder} år.'

print(txt_format())
print(txt_fstring())

tid_format = timeit.timeit(txt_format, number=1000000)
tid_fstring = timeit.timeit(txt_fstring, number=1000000)

print(f'.format() brukte {tid_format:.2f} sekunder.')
print(f'f-string brukte {tid_fstring:.2f} sekunder.')
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

kan derimot være viktig hvis vi jobber med programmer skrevet før Python 3.6 eller på systemer som fortsatt bruker tidligere versjoner av Python.

### Ta med minst dette

Vi kan formatera tekste med `+` operatoren, med `%` operatoren, anvende `str.format()` eller bruke *f-strings*. Vi skal bruke *f-strings*, som ser slik ut: `f"tekst {variabelnavn} mer tekst"`. Verdien til variablene blir flettet inn i teksten med krøllparenteser rundt variabelnavnene. Verdiene kan også formateres ved hjelp av et valgfritt kolon-tegn (`:`) etter variabelnavnet. Denne formateringen har sitt eget, lille minispråk som det ikke er nødvendig å huske i detalj, men det er nyttig å vite at vi for eksempel kan konvertere mellom tallsystmer, begrense antall desimaler, justere teksten, kontrollere antall tegn (width), med mer. For eksempel kan vi bruke `:>10.2f` for et høyrejustert tall med 2 desimaler som opptar 10 tegn. Se på tidligere eksempler eller kom tilbake hit for å få til en bestemt formatering av en tekst.

### Øvingsoppgaver

Dette er greit å kunne for å løse oppgavene til denne bolken:

- Når vi skriver ut tekst, kan vi sette inn en tabulator med `\t` og en ny linje med `\n`.  
`print('Første linje\nAndre linje')` vil skrives på to linjer.
- Når brukeren skal skrive inn et tall, kan vi sjekke om teksten danner et heltall med `str.isnumeric()`.

#### 1.3.1

Bruk kun `print`-setninger og skriv teksten 'Vertikal' med en bokstav på hver linje.

Gjør det samme med en løkke.

En tekst er sekvens med tegn som kan brukes likt som lister i `for`-setninger.

Gjør det samme med kun én `print`-setning og bruken av `\n`.

Gjør det samme med `str.join()`.

```
' + 'join(['a', 'b', 'c']) gir 'a + b + c'  
list('Tekst') gir ['T', 'e', 'k', 's', 't']
```

#### 1.3.2

Skriv et program som oversetter vanlige tall (titallsystemet) til binære tall. Dersom brukeren skriver inn noe som ikke er et heltall, skal det gis en tilbakemelding. Du kan bruke `str.isnumeric()` til å sjekke om teksten danner et heltall.

```
Skriv inn et heltall: 5  
5 i titallsystemet er 101 binært.
```

```
Skriv inn et tall: fem  
fem er ikke et tall.
```

Utvid programmet ved å la det gå i løkke helt til brukeren skriver 'slutt'.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

1.3.3

Skriv ut listen nedenfor slik at tallene kommer under hverandre med komma på samme plass og med 1 desimal.

[1, 2.3, 4.56, 7.890, 12.34, 567.890]

1.0  
2.3  
4.6  
7.9  
12.3  
567.9

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.4 Løkker

#### Bokgens innhold

Itererbare objekter .....	21
Tallrekker med range .....	21
for-løkker .....	22
while-løkker .....	22
for- eller while-løkke? .....	23
break og continue .....	23
Noen praktiske eksempler .....	24
Ta med minst dette .....	24
Øvingsoppgaver .....	24

#### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program

#### Itererbare objekter

Et itererbart objekt (*iterable*) i Python er et objekt som kan levere ett og ett av sine medlemmer om gangen. Det brukes blant annet i gjentakelser (*iterations*), og `for`-løkken tjener nettopp til dette formålet. Den kan brukes til å iterere over elementene i en sekvens, for alle sekvenser er itererbare. Typiske sekvenser er tekststrenger (`str`), lister (`list`) og tallrekker (`range`).

```
for bokstav in 'ord':  
    print(bokstav)
```

ord

```
elever = ['Lise', 'Torill', 'Per', 'Knut', 'Oscar']  
for elev in elever:  
    print(elev)
```

Lise  
Torill  
Per  
Knut  
Oscar

I det første eksempelet gjentas `print`-setningen tre ganger; en gang for hver bokstav i teksten '`ord`'. I det andre eksempelet gjentas `print`-setningen fem ganger; en gang for hvert element (av typen `str`) i listen med variabelnavnet `elever`. Variablene `bokstav` og `elev` mottar neste element i sekvensen for hver gjennomgang av løkken.

#### Tallrekker med range.

`range` (BIF) representerer en uforanderlig sekvens av tall og brukes vanligvis i `for`-løkker når vi ønsker et bestemt antall gjentakelser. `range` kan ta 1 (`stop`), 2 (`start` og `stop`) eller 3 (`start`, `stop` og `step`) argumenter. Sekvensen starter fra `start`-verdien og går til, men inkluderer ikke `stop`-verdien. Hvis vi ikke oppgir `start`, blir 0 brukt. Oppgir vi ikke `step`, blir 1 brukt. `step` kan også være negativ.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
print(list(range(10)))
print(list(range(1,10)))
print(list(range(10,1)))
print(list(range(1,10,2)))
print(list(range(10,1,-2)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[]
[1, 3, 5, 7, 9]
[10, 8, 6, 4, 2]
```

### for-løkker

Det er ikke nødvendig å konvertere `range` til en `list` når vi bruker det i en `for`-løkke. Ovenfor er det kun gjort for å se tallene `range` frembringer. Ellers ville det sett slik ut:

```
print(range(10,1,-2))    range(10, 1, -2)
```

Skal vi skrive ut tallene 10, 8, 6, 4 og 2 med en `for`-løkke kan vi gjøre det slik<sup>7</sup>:

```
runde = 0

for tall in range(10,1,-2):
    runde += 1
    print('Tallet i ' + str(runde) + '. runde er ' + str(tall))
```

```
Tallet i 1. runde er 10
Tallet i 2. runde er 8
Tallet i 3. runde er 6
Tallet i 4. runde er 4
Tallet i 5. runde er 2
```

Først brukes det første tallet i sekvensen, 10, i variabelen `tall` når setningene i løkka utføres. Så hentes det neste tallet, 8, til variabelen `tall` og setningene i løkka utføres på nytt. Slik fortsetter det til det ikke er flere tall igjen i sekvensen. I stedet for `tall` brukes ofte `i` som variabelnavn. Variabelen `runde` er ikke nødvendig for at setningene i løkka utføres. Den viser kun hvordan vi kan telle rundene.

Det er teknisk mulig, men anbefales ikke for lesbarhetens skyld, å ha en setning på samme linje som `for`-setningen: `for i in range(10,1,-2): print(i)`

### while-løkker

Det samme resultatet kan også oppnås med en `while`-løkke. Setningene inni løkka utføres så lenge betingelsen er oppfylt, dvs `teller > 1`. Når vi bruker en `while`-løkke, må vi endre på en variabel, som inngår i `while`-betingelsen. Endringen må skje inni løkka. Hvis ikke, vil løkka fortsette nye runder "i all evighet". Det kalles for en uendelig løkke eller *infinite loop*. Da vil terminalvinduet "henge" eller "fryse" og må lukkes. Det skjer ikke her:

```
teller = 10
while teller>1:
    print(teller)
    teller = teller - 2
```

Rekkefølgen på setningene inni løkka påvirker som vanlig programmet. Endrer vi på den, må vi også endre på startverdien til telleren og betingelsen i `while`-setningen for å oppnå det samme resultatet:

```
teller = 12
while teller>2:
    teller = teller - 2
    print(teller)
```

<sup>7</sup> Denne koden kan forenkles med bruk av `enumerate`, som vil bli forklart senere i avsnittet om [radioknapper og avhukingsbokser](#) i 2.1 Grafisk brukergrensesnitt med tkinter

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Noen ganger kan vi se flere setninger på samme linje skilt med semikolon (:). Selv om det er teknisk mulig, er det bedre for lesbarheten å skrive hver setning på en ny linje. Hvis vi høyreklikker i koden og velger *Format Document* med *autopep8* eller *Black Formatter*, vil setninger som står på samme linje, bli flyttet til hver sin nye linje som ovenfor.

```
teller = 10
while teller>1: print(teller); teller -= 2
```

### for- eller while-løkke?

Vi kan bruke **for**-løkker når vi vet på forhånd hvor mange gjentagelser det blir, og **while**-løkker når vi ikke vet det. Se eksempelet nedenfor med bruk av **break**.

Både **for**- og **while**-løkker kan avsluttes med en valgfri **else**-kodeblokk som kjøres når det det ikke er flere elementer igjen, men den utføres ikke dersom løkken avbrytes med **break**.

### break og continue

Vi kan kontrollere flyten i løkker med **break** og **continue**, enten det er **for**-løkker eller **while**-løkker. Når vi bruker **break** inne i en løkke, vil den umiddelbart avslutte løkken, en eventuell **else**-kodeblokk vil ikke bli utført og en eventuell tellervariabel vil beholde sin nåværende verdi. **continue** brukes til å hoppe over gjenværende kode i den gjeldende gjennomgang og begynner på neste. Det kan gjøre koden mer lesbar enn å ha den gjenværende koden inni i en **if**-kodeblokk.

Hvis vi skal kunne skrive inn så mange tall vi vil og avslutte med å taste Enter, kan vi gjøre det med en tilsynelatende uendelig løkke og **break**:

```
print('Når du er ferdig med å skrive inn tall,')
print('kan du trykke Enter uten å skrive noe.')
antall_tall = 0
while True:
    tekst_inn = input('Skriv inn et heltall: ')
    if tekst_inn == '':
        break
    if tekst_inn.isnumeric():
        tall = int(tekst_inn)
        print('Du skrev inn ' + str(tall))
        antall_tall += 1
    else:
        print(tekst_inn + ' er ikke et heltall.')
print('Du skrev inn ' + str(antall_tall) + ' tall.')
print('Takk for nå')
```

Når du er ferdig med å skrive inn tall,  
kan du trykke Enter uten å skrive noe.  
Skriv inn et heltall: 11  
Du skrev inn 11  
Skriv inn et heltall: 7  
Du skrev inn 7  
Skriv inn et heltall: 5  
Du skrev inn 5  
Skriv inn et heltall: 3  
Du skrev inn 3  
Skriv inn et heltall: 2  
Du skrev inn 2  
Skriv inn et heltall:  
Du skrev inn 5 tall.  
Takk for nå

Samme metodikk kan brukes på eksemplene med å skrive ut tallene 10, 8, 6, 4 og 2. Her bruker vi også kortformen **teller -= 2** som er det samme som **teller = teller - 2**.

```
teller = 10
while True:
    print(teller)
    teller -= 2
    if teller < 2:
        break
```

Vi tar også med et eksempel på bruk av **continue** og **else**. Dersom vi hadde brukt **break** i stedet for **continue**, ville kun **10** og **8** bli skrevet ut, heller ikke **else**-blokken og **i** ville vært **6** etterpå.

```
# Hopp over 6
for i in range(10, 1, -2):
    if i == 6:
        continue
    print(i)
else:
    print('Ferdig')
    print(f'i er nå {i}.')
```

10
8
4
2
Ferdig
i er nå 2.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Noen praktiske eksempler

Hvis vi vil skrive ut en liste over flere rader med et spesifisert antall kolonner, kan vi bruke en teller og `%` operatoren sammen med en kombinasjon av `for`- og `if`-setninger. Hver gang telleren er delelig med antall kolonner, setter vi inn en ny linje. Her skriver vi ut alle oddetall mellom 1 og 100 over 15 kolonner.

```
tabell = range(1,101,2) # 1,3,5,...  
antall_kolonner = 15  
teller = 0  
for tall in tabell:  
    if teller % antall_kolonner == 0:  
        print() # ny rad  
    teller += 1  
    print(f'{tall: 4}', end='')
```

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29
31	33	35	37	39	41	43	45	47	49	51	53	55	57	59
61	63	65	67	69	71	73	75	77	79	81	83	85	87	89
91	93	95	97	99										

I `f`-strenget angir vi en bredde på 4 tegn og mellomrom som fylltegn. I `print`-setningen overstyrer vi ny linje til slutt, som er standard, med en tom streng.

Vi kan også ha en `for`-løkke inni en `for`-løkke, eksempelvis for å skrive ut en begrenset gangetabell. Den ytre `for`-løkka angir raden, og den indre `for`-løkka angir kolonnen.

```
for rad in range(1, 6):  
    print()  
    for kolonne in range(1, 6):  
        print(f'{rad*kolonne: 4}', end='')
```

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

### Ta med minst dette

Når vi skal gjenta en kodeblokk flere ganger, kan vi bruke `for`- eller `while`-løkker. En `for`-løkke kjører et angitt antall ganger, som vi vet på forhånd. En `while`-løkke kjører så lenge en bestemt betingelse er oppfylt. For at en `while`-løkke skal stoppe, må vi endre en variabel som inngår i betingelsen inne i løkken. Hvis ikke, risikerer vi at løkken aldri stopper. Noen objekter er også sekvenser, som `str`, `list` og `range`. De er itererbare og kan brukes i `for`-løkker. `range` starter fra et gitt startpunkt (0 som standard) og går opp til, men ekskluderer, stoppverdien. Vi kan kontrollere flyten i løkker med `break` og `continue`.

### Øvingsoppgaver

#### 1.4.1

Forklar hva som skjer her:

```
for i in range(1,4):  
    print('i: '+ str(i))  
    for j in range(1,3):  
        print('i: '+ str(i) + ' og j: ' + str(j))
```

#### 1.4.2

Lag et program som skriver ut alle partallene fra 100 og ned til og med 80.

```
100 98 96 94 92 90 88 86 84 82 80
```

#### 1.4.3

Lag et program som skriver ut de 10 første kubikktallene på én linje (1, 8 ,27...).

```
1 8 27 64 125 216 343 512 729 1000
```

Vurderingerseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

**1.4.4**

Lag kolonne- og rad-overskrifter til den begrensede gangetabellen:

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

**1.4.5**

Lag et program som skriver ut alle [Fibonaccitallene](#) under 1000 i fem kolonner. Det første tallet er **0**. Det andre er **1**. Det tredje er summen av de to foregående, **0+1=1**. De fjerde er summen av de to foregående tallene, **1+1=2**, og slik fortsetter det: **1+2=3**, **2+3=5**, **3+5=8**, osv.:

**0, 1, 1, 2, 3, 5, 8, 13, 21, ...**

Fibonaccitallene under 1000:				
0	1	1	2	3
5	8	13	21	34
55	89	144	233	377
610	987			

## 1.5 Lister

### Bolkens innhold

Opprette, indeksere, og slette lister .....	26
Legge elementer til en liste .....	27
Fjerne elementer fra en liste.....	27
Søke etter elementer i en liste.....	28
Sortere og reversere lister.....	28
Kopiere lister .....	28
Statistiske funksjoner og analyse av lister.....	29
Flerdimensjonal lister .....	29
Listebygging ( <i>List comprehension</i> ) .....	30
Unngå sletting i en vanlig for-løkke.....	30
Ta med minst dette.....	31
Øvingsoppgaver.....	31

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program

### Opprette, indeksere, og slette lister

En liste er en samling elementer ordnet i en bestemt rekkefølge. Lister kan opprettes med klammeparenteser `[]` eller ved å bruke den innebygde funksjonen `list()`. For å få tilgang til eller endre elementer i listen, bruker vi indekser inni `[]`. Indeksering starter ved `0`, som betyr at det første elementet har indeks `0`, det neste har indeks `1`, og så videre. Vi kan benytte den innebygde funksjonen `len()` for å finne antallet elementer i en liste. Dermed kan det siste elementet nås med `len(listen) - 1` eller ved å bruke indeksen `-1`. Nest siste element kan nås med `-2`, osv. Å forsøke å få tilgang til en indeks utenfor listen (dvs. større enn `len(listen) - 1` eller mindre enn `-len(listen) - 1`) vil resultere i en `IndexError: list index out of range`.

```
venner = ['Ole', 'Donald', 'Doffen']

print(type(venner))
print(len(venner))
print(venner, '\n')

print(venner[0])
print(venner[1])
print(venner[-1], '\n')

venner[1] = 'Dole'
print(venner)
```

```
<class 'list'>
3
['Ole', 'Donald', 'Doffen']

Ole
Donald
Doffen

['Ole', 'Dole', 'Doffen']
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

For å hente ut deler av en liste benytter vi [`<start>: <stopp>:<step>`]. Et utsnitt (*slice*) starter fra og med startindeksen og går opp til, men ikke inkludert sluttindeksen. Uten startindeks starter den på begynnelsen av listen, og uten sluttindeks går den til slutten av listen.

```
venner = ['Ole', 'Dole', 'Doffen', 'Donald']
print(venner)
print(venner[1:3])
print(venner[:2])
print(venner[2:])
print(venner[1::2])
print(venner[:])
```

```
['Ole', 'Dole', 'Doffen', 'Donald']
['Dole', 'Doffen']
['Ole', 'Dole']
['Doffen', 'Donald']
['Dole', 'Donald']
['Ole', 'Dole', 'Doffen', 'Donald']
```

Å slette en liste kan gjøres på flere måter. Ved å bruke `del` på en liste, som i `del listen`, fjernes hele listen. Ved å bruke `listen.clear()` beholdes listen, men alle elementene fjernes. Ved å tildele en tom liste til en variabel (f.eks. `listen = []`), erstatter vi innholdet i variabelen med en ny, tom liste. Den opprinnelige listen vil bli håndtert av systemets *garbage collector* hvis ingen referanser peker til den.

### Legge elementer til en liste

`append()` legger til et element på slutten av listen, `extend()` legger til flere elementer (fra en annen liste) på slutten av liste, og `insert()` setter inn et element på en bestemt plass i listen,

```
venner = ['Ole', 'Dole', 'Doffen']
print(venner)

venner.append('Hetti')
print(venner, '\n')

venner.extend(['Netti', 'Donald'])
print(venner, '\n')

venner.insert(4, 'Letti')
print(venner)
```

```
append:
['Ole', 'Dole', 'Doffen']
['Ole', 'Dole', 'Doffen', 'Hetti']
extend:
['Ole', 'Dole', 'Doffen', 'Hetti', 'Netti', 'Donald']
insert:
['Ole', 'Dole', 'Doffen', 'Hetti', 'Letti', 'Netti', 'Donald']
```

### Fjerne elementer fra en liste

`remove()` fjerner første forekomst av et element fra listen (gitt verdien). Dersom vi prøver å fjerne et element som ikke finnes i listen, får vi `ValueError: list.remove(x): x not in list`.

`pop()` fjerner og returnerer et element fra en bestemt plass i listen (gitt indeksen). Hvis vi ikke oppgir en indeks, fjernes det siste elementet fra listen. `del` fjerner et element gitt en indeks eller et indeksintervall. Dersom indeksen ikke er i listen, får vi `IndexError: list assignment index out of range`.

```
print('remove:')
venner = ['Ole', 'Dole', 'Doffen', 'Donald', 'Dolly']
print(venner)
venner.remove('Donald')
print(venner)
print('pop:')
navn = venner.pop(1)
print(navn)
print(venner)
print('del:')
del venner[1:]
print(venner)
```

```
remove:
['Ole', 'Dole', 'Doffen', 'Donald', 'Dolly']
['Ole', 'Doffen', 'Dolly']
pop:
Dole
['Ole', 'Doffen', 'Dolly']
del:
['Ole']
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Søke etter elementer i en liste

`index()` finner indeksen til et bestemt element i listen. Som argumenter oppgir verdien vi søker etter, og vi kan også ha med en startindeks for søket, som er nyttig når det er flere forekomster av samme verdi. Hvis elementet ikke finnes i listen, returneres `ValueError: '...' is not in list`. Vi kan bruke `in` for å sjekke om et element eksisterer i listen, og `count()` for å telle hvor mange ganger et bestemt element forekommer.

```
venner = ['Ole', 'Dole', 'Doffen', 'Dole', 'Dolly']
print(venner)
navn = 'Doffen'
if navn in venner:
    indeks = venner.index(navn)
    print(f'Første forekomst av {navn} er i posisjon {indeks}')
else:
    print(f'{navn} forekommer ikke i venner-listen.')
navn = 'Dole'
antall = venner.count(navn)
print(f'{navn} forekommer {antall} ganger i venner-listen.)
```

```
['Ole', 'Dole', 'Doffen', 'Dole', 'Dolly']
Første forekomst av Doffen er i posisjon 2
Dole forekommer 2 ganger i venner-listen.
```

### Sortere og reversere lister

`sort()` sorterer elementene i listen. `sorted()` returnerer en ny, sortert liste (og beholder den originale listen uendret). `reverse()` snur rekkefølgen på elementene i listen.

```
print('sort:')
venner = ['Ole', 'Dole', 'Doffen']
print(venner)
venner.sort()
print(venner)
print('sorted:')
venner = ['Ole', 'Dole', 'Doffen']
print(sorted(venner))
print(venner)
print('reversed:')
print(venner)
venner.reverse()
print(venner)
```

```
sort:
['Ole', 'Dole', 'Doffen']
['Doffen', 'Dole', 'Ole']
sorted:
['Doffen', 'Dole', 'Ole']
['Ole', 'Dole', 'Doffen']
reversed:
['Ole', 'Dole', 'Doffen']
['Doffen', 'Dole', 'Ole']
```

### Kopiere lister

Hvis vi har en liste `a` og tildeler denne til en ny variabel `b` med `b = a`, vil begge variablene peke på den samme listen. Dette betyr at endringer vi gjør med `b` også vil påvirke `a`, og omvendt, fordi de peker på den samme listen. Dersom vi ønsker to like, men uavhengige lister, kan vi benytte `copy()`. Da vil endringer i `c` ikke påvirke `a`. Ved å bruke `id()`-funksjonen ser vi at `a` og `b` har samme id og peker til samme objekt, mens `c` har en annen id, selv om innholdet er det samme som i `a` og `b`.

Når vi vil sjekke om to variabler peker på samme objekt, bruker vi `is`-operatoren. For å sjekke om to lister har samme innhold, bruker vi `==`-operatoren. I vårt eksempel vil `a is b` gi `True` fordi begge peker på samme liste, mens `a is c` vil gi `False`, selv om `a == c` vil gi `True` fordi innholdet i

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

listene er det samme.

```
a = [1,2]
print(f'a: {a}')
b = a
print(f'a is b: {a is b}')
print(f'a == b: {a == b}')
c = a.copy()
print(f'a is c: {a is c}')
print(f'a == c: {a == c}')
b[0]=3
c[1]=4
print(f'b: {b}')
print(f'c: {c}')
print(f'a: {a}')
print(f'id a = {id(a)})')
print(f'id b = {id(b)})')
print(f'id c = {id(c)})')
```

```
a: [1, 2]
a is b: True
a == b: True
a is c: False
a == c: True
b: [3, 2]
c: [1, 4]
a: [3, 2]
id a = 1361229943424
id b = 1361229943424
id c = 1361231980736
```

### Statistiske funksjoner og analyse av lister

Det finnes innebygde funksjoner for å finne **sum**, **minimum** og **maksimum** til lister med tall.

```
tall = [5,3,8,11,9,3,2]
print(f'\n\n\tall: {tall}')

print('\nSum, minst og størst:')
print(f'Sum: {sum(tall)}')
print(f'Min: {min(tall)}')
print(f'Max: {max(tall)}')
```

```
tall: [5, 3, 8, 11, 9, 3, 2]
Sum, minst og størst:
Sum: 41
Min: 2
Max: 11
```

Hvis vi skriver **mean**, **median** og **mode** for å beregne gjennomsnitt, median og typetall, vil IntelliSense foreslå automatisk `import from statistics import mean, median, mode`.

```
from statistics import mean, median, mode
print('\nGjennomsnitt, median og typetall:')
print(f'Mean: {mean(tall)}')
print(f'Median: {median(tall)}')
print(f'Mode: {mode(tall)}')
```

```
Gjennomsnitt, median og typetall:
Mean: 5.857142857142857
Median: 5
Mode: 3
```

### Flerdimensjonal lister

Flerdimensjonale lister er lister som inneholder andre lister. En vanlig type er todimensjonale lister, som en tabell med rader og kolonner. Her er et eksempel med temperaturmålinger fra forskjellige byer. Radene representerer ulike dager, og kolonnene representerer henholdsvis Oslo, Bergen og Trondheim.

For å skrive ut denne tabellen kan vi bruke en for-løkke inni en annen (*nested*). For å hente et bestemt element angir vi først raden og deretter kolonnen, som `tabell[rad][kolonne]`.

En vanlig ("grunn"/*shallow*) `copy()` av listen kopierer ikke verdiene i de indre liste-elementene, men bare referansen til disse listene. Dette betyr at hvis vi endrer en verdi i kopien, vil den tilsvarende verdien i den originale listen også endres, og omvendt. Med `deepcopy()` får vi en ("dyp"/*deep*) kopi av alle verdiene, slik at endringer i originalen ikke påvirker kopien, og omvendt.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
import copy

temperaturer = [
    [22, 20, 18], # Dag 1: Oslo, Bergen og Trondheim
    [21, 20, 20], # Dag 2
    [17, 17, 17], # Dag 3
]

for dag in temperaturer:
    for by in dag:
        print(f"{by:>4}", end="")
    print()

grunn_kopi = temperaturer.copy()
dyp_kopi = copy.deepcopy(temperaturer)
temperaturer[1][2] = 19 # Endrer temp i Trondheim dag 2

print(temperaturer)
print(grunn_kopi)
print(dyp_kopi)
```

```
22 20 18
21 20 20
17 17 17
[[22, 20, 18], [21, 20, 19], [17, 17, 17]]
[[22, 20, 18], [21, 20, 19], [17, 17, 17]]
[[22, 20, 18], [21, 20, 20], [17, 17, 17]]
```

### Listebygging (*List comprehension*)

Listebygging vil litt forenklet si å lage lister basert på andre lister. Dette er litt forenklet fordi det gjelder ikke bare lister, men alle objekter vi kan gå gjennom med en for-løkke. Dette kan vi for eksempel bruke til å transformere tallene fra 1 til 10 til kvadrattall, eller vi kan filtrere ut alle tallene mellom 1 og 100 som inneholder sifferet 2, og mye annet, som å gjøre en matrise om til en endimensjonal tabell. Alt i alt er dette en kompakt, elegant og funksjonell måte for å bygge og transformere lister, selv om det alltid går an å skrive om en listebygging til en for-løkke.

```
kvadrater = [x**2 for x in range(1, 11)]
print("Kvadrater:\n", kvadrater)

tall = [x for x in range(1, 101) if "2" in str(x)]
print("\nTall med siffer 2:", end="")
for i in range(0, len(tall)):
    if i % 10 == 0:
        print()
    print(f"{tall[i]:>3}", end="")
print()

matrise = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print("\nMatrise:")
for rad in matrise:
    print(rad)
flat_liste = [tall for rad in matrise for tall in rad]
print("Flat liste:\n", flat_liste)
```

```
Kvadrater:
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Tall med siffer 2:
 2 12 20 21 22 23 24 25 26 27
 28 29 32 42 52 62 72 82 92

Matrise:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Flat liste:
 [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

### Unngå sletting i en vanlig for-løkke

Å slette elementer fra en liste inni en for-løkke kan forårsake uforutsigbare resultater. Når vi fjerner et element, forskyves indeksene til de gjenværende elementene og lengden på listen endres. Dette kan føre til at vi hoppet over noen elementer, eller at programmet avbrytes med

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

IndexError: list index out of range, som her hvor vi prøver å fjerne alle 3-tall fra listen.

```
# FEIL
tall = [1,2,3,3,3,4,3,3,3,5,3,3,6,3]
for t in tall:
    if t == 3:
        tall.remove(t)
print(tall)

# index out of range
# tall = [1,2,3,3,3,4,3,3,3,5,3,3,6,3]
# for i in range(len(tall)):
#     if tall[i] == 3:
#         tall.pop(i)
# print(tall)
```

[1, 2, 4, 5, 3, 3, 6, 3]

For å unngå disse problemene kan vi eksempelvis gå gjennom løkken bakfra, bruke listebygging og lage en ny liste uten de elementene som skal fjernes, eller bruke en `while`-løkke med betingelse for sletting.

```
# RIKTIG
tall = [1,2,3,3,3,4,3,3,3,5,3,3,6,3]
for i in range(len(tall)-1,-1,-1):
    if tall[i] == 3:
        tall.pop(i)
print(tall)

tall = [1,2,3,3,3,4,3,3,3,5,3,3,6,3]
tall = [x for x in tall if x != 3]
print(tall)

tall = [1,2,3,3,3,4,3,3,3,5,3,3,6,3]
while 3 in tall:
    tall.remove(3)
print(tall)
```

[1, 2, 4, 5, 6]  
[1, 2, 4, 5, 6]  
[1, 2, 4, 5, 6]

### Ta med minst dette

En liste er en sammensatt datatype som samler andre objekter i en ordnet rekkefølge. Vi oppretter lister med `= []`, `= list()` eller `= [objekt, objekt, ...]`. Hvert element kan nås via en indeks som begynner på 0; for eksempel gir `liste[1]` oss det andre objektet. Lister har metoder for å legge til (`append`, `extend` og `insert`) og fjerne (`remove`, `pop`) objekter. Det finnes dessuten metoder for å søke (`index`), sortere (`sort`) og reversere (`reverse`) lister, samt statistiske metoder for lister med tall. Husk at det er fallgruver ved sammenligning av lister, ved kopiering av flerdimensjonale lister og ved sletting av objekter fra en liste inni en for-løkke.

### Øvingsoppgaver

#### 1.5.1

Gitt listen `tall = [10, 20, 30, 40, 50]`

- Skriv ut det første og siste elementet.
- Skriv ut listen i reversert rekkefølge.

**Hint:** Bruk `[::-1]` for å skrive ut listen i omvendt rekkefølge uten å endre den opprinnelige listen.

- Endre det tredje elementet til 35 og skriv ut hele listen igjen.

Vurderingsseksempler

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

#### 1.5.2

Gitt listen `frukt = ['eple', 'appelsin', 'banan']`

- a) Legg til 'kiwi' på slutten av listen og skriv den ut.
- b) Finn indeksen til 'appelsin' og erstatt 'appelsin' med 'plomme'
- c) Fjern 'banan' fra listen ved å bruke verdien.

#### 1.5.3

Gitt listen `bokstaver = ['a', 'b', 'c', 'd', 'e', 'f', 'g']`. Bruk deler av en liste (*slices*) og

- a) Skriv ut de første tre elementene.
- b) Skriv ut de tre midterste elementene.
- c) Skriv ut listen fra det tredje til det siste elementet.

#### 1.5.4

Gitt listen `tall = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

- a) Bruk en `for`-løkke for å skrive ut hvert element multiplisert med 10, på én linje.
- b) Bruk listebygging (*list comprehension*) til å lage en ny liste som inneholder alle tallene i tall økt med 5. Skriv ut den nye listen

#### 1.5.5

Gitt listen `tall = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]`

Skriv et program som fjerner annet hvert element fra listen slik at den blir `[10, 30, 50, 70, 90]`.

#### 1.5.6

Opprett en todimensjonal liste (8x8) som representerer et sjakkbratt, hvor hvert element er enten 'svart' eller 'hvit'. Skriv ut listen rad for rad hvor rutene er justert kolonnevis.

#### 1.5.7

Lag et program som ber brukeren skrive inn heltall som du legger inn i en liste. For hvert tall som legges inn, skal summen av alle tallene i listen skrives ut.

Når brukeren trykker Enter uten å skrive inn et heltall, skal programmet skrive ut listen og avslutte.

Før programmet avsluttet skal de største og laveste tallet fjernes fra listen samtidig som listen og gjennomsnittet av tallene skrives ut.

```
Skriv inn et heltall: 3
Summen er nå 3
Skriv inn et heltall: 5
Summen er nå 8
Skriv inn et heltall: 8
Summen er nå 16
Skriv inn et heltall: 2
Summen er nå 18
Skriv inn et heltall: 11
Summen er nå 29
Skriv inn et heltall: 4
Summen er nå 33
Skriv inn et heltall:
[3, 5, 8, 2, 11, 4]
[3, 5, 8, 4]
Gjensomsittet uten min og maks er 5
```

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

1.5.8

Bruk algoritmen [Eratosthenes' sil](#) til å skrive ut alle primtallene under 100 i fem kolonner:

- Lag først en liste med tallene fra 2 til 100.
- Bruk det først tallet i tabellen (2) og fjern alle tallene i tabellen som er delelig med dette første tallet bortsett fra tallet selv.
- Bruk det neste tallet i tabellen (3) og fjern alle tallene i tabellen som er delelig med dette neste tallet bortsett fra tallet selv.
- Slik fortsetter du inntil du har testet tall mindre eller lik roten av 100, dvs 10.
- Skriv ut de gjenværende tallene (primtallene) i fem kolonner som vist under.

Primtall under 100:				
2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97

Vurderingsseksemplar

## 1.6 Filer

### Bolkens innhold

Grunnleggende om filer.....	34
Filtilgang .....	35
Filoperasjoner .....	36
Ta med minst dette.....	37
Øvingsoppgaver.....	37

### Kompetansemål:

- gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

### Grunnleggende om filer

Elektroniske dokumenter, digitale bilder, dataprogrammer, osv. lagres i **filer** på en datamaskin. De primære lagringsenhetene er platelager som HDD eller SSD, men data kan også lagres eksternt på minnepinner eller minnekort. Hvor og hvordan vi lagrer data innvirker på sikringen av dem, se [KM10](#). Vi skiller gjerne mellom programfiler og datafiler basert på innholdet selv om de er tilsynelatende like. Alle filer består av 0-er og 1-ere (*bit = binary digit*). Disse er gruppert i **byter**, som består av 8 bit. Operativsystemet holder styr på hvor filen begynner (den første byten) og hvor filen slutter (den siste byten).

Filer har filnavn som består av <**navn**>.<**filtype**>. Både <navn> og <filtype> kan vi velge selv. Noen spesialtegn er ikke tillatt i navnene. Selv om vi står fritt til å velge navn på filtypen, bør vi følge vanlig standard: **.app** for applikasjonspakker i macOS, **.exe** eller **.com** for programfiler i Windows, **.docx** for Word filer, **.jpg** for bildefiler (av den typen), **.html** for websider, osv. Se eksempelvis [fileinfo.com](#). Operativsystemet forbinder filtypene med bestemte programmer, men brukere kan også endre disse standardinnstillingene. Hvis vi dobbeltklikker på en **.docx**-fil, starter Word og leser inn datafilen. Hvis vi dobbeltklikker på en **.exe**-fil, tror operativsystemet det er et program og prøver å kjøre det.

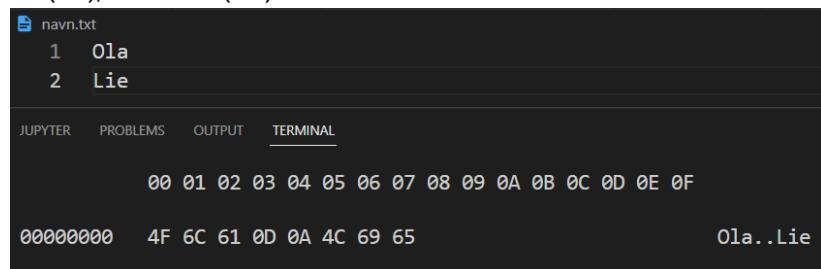
Mange filtyper har dessuten en filsignatur (magisk nummer) først i fila. Dette kan programmene bruke til å gjenkjenne formatert (eller filtypen) til innholdet. For eksempel begynner en **.jpg**-fil ofte med **FF D8**, mens **.mp3**-filer typisk starter med **49 44 33 [ID3]**. For å utforske slike signaturer, kan vi se på denne [listen](#) som gir en oversikt over ulike filsignaturer. Vi kan se **hex**-kodene til filens innhold i Terminalvinduet i VS Code med **format-hex**-kommandoen, forutsatt at vi bruker PowerShell som er standard. Det virker ikke med Ledetekst (cmd). Vi kan begrense oss til den første linjen med **format-hex <filnavn> | Select-Object -First 1**. I macOS kan vi bruke **hexdump -C -n N <filnavn>** hvor **N** er antall bytes vi ønsker å se.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Rene tekstfiler (som **.txt**, **.csv** og **.json**) mangler vanligvis filsignatur. Når slike tekstfiler inneholder strukturerte data, omtales de ofte som "flate filer". En linje kan sammenliknes med en rad eller post i en database. En **.txt**-fil kan ha kolonner eller felt på fastsatte posisjoner, mens en **.csv**-fil (*comma-separated values*) skiller verdiene med komma, og da behøver ikke feltene stå i samme posisjon. **.json**-formatet skal vi lære om senere.

I VS Code kan vi opprette en fil som vi kaller **navn.txt** og skrive fornavnet vårt på første linje og etternavnet på andre linje. Deretter lagrer vi endringene med **File Save** (eller **Ctrl+S**) og skriver **format-hex navn.txt** i terminalvinduet. Da får vi se **hex**-koden til tegnene, og kan legge merke til at det står **OD OA** mellom første og andre linje. Dette ble satt inn da vi trykket Enter-tasten. **OD** er hex for kontrolltegnet **CR (Carriage Return)**. Uttrykket stammer fra gamle skrivemaskiner hvor vi måtte skyve vogna tilbake med en spak da vi kom til slutten av en linje. **OA** er hex for **LF (Line Feed)**. Spaken kunne også brukes til å rulle vlsen en eller flere linjer fram. I macOS brukes kun **LF (OA)**, uten CR (**OD**)



### Filtilgang

Når vi skal lese data fra en fil i Python, bruker vi den innebygde funksjonen **open()**. Selv om filen vi ønsker å åpne ligger i samme mappe som programmet vårt, kan vi støte på problemer. Dette kan spesielt være tilfelle hvis vi har åpnet en annen mappe i VS Code via **File** > **Open Folder**.... For å omgå dette problemet, kan vi enten bruke **File** > **Open Folder**... for å åpne mappen hvor programmet ligger, eller justere innstillingene i VS Code. Klikk på tannhjulet nederst til venstre, velg **Settings**, søk deretter etter **python.terminal** og huk av for **Execute in File Dir**.

**Python** > **Terminal: Execute In File Dir**

When executing a file in the terminal, whether to use execute in the file's directory, instead of the current open folder.

For å være på den sikre siden og alltid kunne nå riktig fil, uansett hvilken mappe vi jobber fra, kan vi bruke **pathlib**. Ved å bruke **Path(\_\_file\_\_).resolve().parent** får vi stien til mappen hvor programmet ligger. Selv om **Path** tillater oss å skjøte stien og filnavnet med **/**, kan dette være forvirrende siden Windows benytter **\**. Derfor kan det være fornuftig å holde seg til metoden **joinpath**. Hvis vi har en datafil med filbanen **data/txt/filnavn.txt**, hvor **data**-mappen ligger side om side med **src**-mappen, kan vi nå datafilen fra programmet med **Path(\_\_file\_\_).resolve().parent.parent.joinpath('data', 'txt', 'filnavn.txt')**.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'PROGS' containing 'pathlib', 'data', 'txt', and 'src'. Inside 'src', there is a file named 'program.py'. The 'program.py' tab is active in the main editor area, displaying the following code:

```
1 from pathlib import Path
2
3 print(f'Mappesti til programmet: {Path(__file__).resolve().parent}')
4 fil = Path(__file__).resolve().parent.parent.joinpath("data", "txt", "filnavn.txt")
5 print(f'Filbane: {fil}')
6 with fil.open() as f:
7     print(f.read())
```

Below the editor, the terminal window shows the output of running the script:

```
Mappesti til programmet: C:\Users\ola\OneDrive\Desktop\progs\pathlib\src
Filbane: C:\Users\ola\OneDrive\Desktop\progs\pathlib\data\txt\filnavn.txt
Du fant meg jo.
```

I Jupyter fungerer ikke `__file__` fordi koden kjøres interaktivt. Vi kan imidlertid bruke `Path(os.getcwd()).parent.joinpath("data", "txt", "filnavn.txt")` forutsatt at vi ikke endrer arbeidskatalogen inni notatblokken.

### Filoperasjoner

Vi kan få hele innholdet som tekst (`read`), hele innholdet som en liste (`readlines`) eller linje for linje (`readline`).

The screenshot shows a Jupyter notebook cell with the following Python code:

```
1 with open('navn.txt') as fil:
2     data = fil.read()
3     print('read():')
4     print(data)
5
6     fil.seek(1)
7     data = fil.read(2)
8     print('\nseek(1); read(2):')
9     print(data)
10
11    fil.seek(0)
12    linjer = fil.readlines()
13    print('\nreadlines():')
14    print(linjer)
15
16    fil.seek(0)
17    print('\nreadline():')
18    print(fil.readline().lower(), end='')
19    print(fil.readline().lower())
20
21    fil.seek(0)
22    print('\nfor linje in fil:')
23    for linje in fil:
24        print(linje.upper(), end='')
```

To the right of the code, the output of the cell is shown:

```
read():
Ola
Nordmann

seek(1); read(2):
la

readlines():
['Ola\n', 'Nordmann']

readline():
ola
nordmann

for linje in fil:
OLA
NORDMANN
```

Når vi bruker `with open()` i stedet for `fil=open()`, slipper vi å lukke filen med `fil.close()`. Det er kjekt dersom noe skulle feile så vi aldri kommer til `fil.close()`. Filpekeren, posisjonen vi leser fra, flytter seg etter hvert som vi leser data fra filen. `seek()` flytter filpekeren til oppgitt posisjon. `read(2)` leser 2 tegn. `print()` avsluttes som standard med ny linje. Det kan vi undertrykke med `end=""` siden teksten allerede inneholder `\n`.

Når vi skal skrive til en fil, må vi velge `mode='w'`. Hvis filen ikke finnes, blir den opprettet. Hvis den finnes, blir den overskrevet. Dersom vi ønsker å legge noe til en eksisterende fil, må vi bruke `mode='a'`. Hvis teksten inneholder norske bokstaver (æ, ø og å), må vi også bruke `encoding='utf-8'`.

Vurderingssempolar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
26  with open('norsk.txt', mode='w', encoding='utf-8') as fil:  
27      fil.write('1: EN\n2: TO\n3: TRE\n')  
28      fil.write('æ ø å Æ Ø Å')
```

```
norsk.txt  
1 1: EN  
2 2: TO  
3 3: TRE  
4 æ ø å Æ Ø Å
```

```
format-hex norsk.txt
```

	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	
00000000	31 3A 20 45 4E 0D 0A 32 3A 20 54 4F 0D 0A 33 3A	1: EN..2: TO..3:
00000010	20 54 52 45 0D 0A C3 A6 20 C3 B8 20 C3 A5 20 C3	TRE..Ã  Ã, Ã¥ Ã
00000020	86 20 C3 98 20 C3 85	Ã

Hvis vi sjekker en [utf-8 tabell](#), kan vi se at den siste bokstaven Å er lagret som **C3 85** (hex) i to bytes i motsetning til den første bokstaven 1, **31** (hex), som er lagret i én byte.

U+00C5	Å	c3 85	&#xC5;	Å	LATIN CAPITAL LETTER A WITH RING ABOVE
--------	---	-------	--------	---	--

### Ta med minst dette

Mye av informasjonen i denne bolken er av generell karakter og bør allerede være kjent. Mens detaljer som heksadesimale koder og filsigraturer kan være nyttig i spesielle sammenhenger, er det viktigere å ha en grunnleggende forståelse av vanlige filoperasjoner som `open`, `read`, `readlines`, `readline` og `close`. Når vi jobber med norsk tekst som kan inneholde æ, ø og å, må vi åpne filen med riktig tegnsett, for eksempel ved å bruke `open()` med `encoding='utf-8'`. I Windows brukes CR+LF (`\r\n`) for linjeskift i tekstmater, mens macOS og Linux bruker kun LF (`\n`). Dette kan resultere i forskjeller i hvordan filer leses og vises på tvers av disse plattformene. Senere skal vi forenkle filhåndteringen ved å bruke biblioteker som `csv`, `json` og ikke minst `pandas`. Uansett, må vi huske å lukke en fil etter vi har åpnet den. Hvis ikke, risikerer vi blant annet at endringer går tapt, at filen blir ødelagt, eller at vi ikke får tilgang til den igjen. Dette kan unngås ved å bruke `with`-blokker.

### Øvingsoppgaver

#### 1.6.1

- Ta utgangspunkt i oppgave 1.5.1. I stedet for å legge heltallene inn i en liste, skal de lagres på hver sin linje i en `.txt`-fil som heter `heltall.txt`. Legg inn mellom 10 og 15 verdier. Kall programfilen `skriv_til_fil.py`.

Skriv `notepad heltall.txt` i Terminalvinduet i VS Code og sjekk innholdet i filen med Notisblokk-programmet.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

- b) Bytt `heltall.txt` med en annen elev. Skriv et program som leser innholdet i filen og
- Skriver ut hvor mange tall filen inneholder
  - Skriver ut summen av tallene
  - Skriver ut alle tallene på én linje.

Kall programfilen `les_fra_fil.py`.

#### 1.6.2

- a) De tre filene `fil_1.xxx`, `fil_2.xxx` og `fil_3.xxx` har hver sin ukjente filtype, som er enten `.txt` (tekstfil), `.jpg` (bildefil) eller `.mp3` (lydfil). Skriv et program som
- Lar brukeren skrive inn et filnavn
  - Sjekker de første bytene i filen og skriver ut om filsignaturen er `.jpg`, `.mp3` eller `ukjent`.

Kall programmet `sjekk_signatur.py` og finn filtypen til de tre utleverte filene. Husk å åpne filene i *binary mode* med `mode = 'rb'`

- b) Dobbelt-klikk på filene i filutforskeren.  
Hva skjer? Forklar.

Gi filene nytt navn med riktig filtype. Dobbelt-klikk på filene i filutforskeren.  
Hva skjer? Forklar.

Gi filene det gamle navnet med filtype `xxx`. Prøv å åpne filene i en nettleser.  
Hva skjer? Forklar.

Alle filene inneholder et kodeord.  
Hvilken fil inneholder hvilket kodeord?

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.7 Feil

#### Bolkens innhold

Syntaks og Semantikk.....	39
Typer av Feil i Programmering .....	39
Debugging og Logiske Feil .....	40
Testprogrammer og Assert-setninger .....	41
Ta med minst dette.....	42
Øvingsoppgaver.....	42

#### Kompetansemål:

- vurdere og bruke strategier for feilsøking og testing av programkode
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

#### Syntaks og Semantikk

**Syntaks** handler om hvordan ord settes sammen til setninger og om riktig bruk av stavemåte og grammaikk. **Semantikk** handler om setningenes betydning, hvordan vi tolker dem. En setning kan ha riktig syntaks, men ikke ha noen mening: "Den våte kluten er alltid tørr og plystrer muntert." I Python og andre programmeringsspråk skiller vi også mellom syntaks og semantikk. Hvis vi ikke følger reglene, oppstår **feil**, og det er nyttig å kunne skille mellom ulike typer feil. Det finnes også en tredje type feil: **Logiske feil** oppstår når programmet kjører uten feil, men gir galt resultat. Skillet mellom semantiske og logiske feil kan være uklar.

```
print('Hallo!'          # Syntaksfeil: manglende sluttparentes

total = 3 + 'feil'     # Semantisk feil: meningsløst å addere et tall med en tekst.

alder = 4
if alder >= 18:
    print('umyndig')
else:
    print('myndig')   # Logisk feil: fireåringer er ikke myndige
```

#### Typer av Feil i Programmering

Det er lettare å skille mellom når feilene oppstår: **kompileringsfeil** og **kjøretidsfeil**. Kompileringsfeil oppstår når høynivå programkode oversettes til lavnivå maskinkode som datamaskinen kan utføre. Syntaksfeil gir kompileringsfeil, slik at programmet ikke vil kjøre. Det er litt annerledes med Python fordi programkoden blir oversatt til bytekode som kjøres av fortolkeren (*the Python interpreter*), som er en virtuell maskin. Et Python-program med syntaksfeil vil ikke kjøre forbi feilen og vil rapportere en **SyntaxError** når det forsøker å oversette den feilaktige koden til bytekode.". Disse feilene er vanligvis lette å finne og rette, spesielt med hjelp fra **Pylance**-utvidelsen i VS Code, som forbedrer IntelliSense med avansert

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

språkstøtte og sjekker koden mens vi skriver. En rød krøllet strek under koden indikerer en feil, og når vi fører muspekeren over denne koden, dukker det opp mer hjelp til å rette feilen. Vi vil få den samme beskjeden dersom vi forsøker å kjøre programmet.

A screenshot of the VS Code interface. On the left, there is a dark sidebar with a small preview of the code. On the right, the main editor window shows the following Python code:

```
print('Hallo!')
```

A red squiggly underline is under the opening parenthesis '(', indicating a syntax error. A tooltip above the cursor says "()" was not closed Pylance. Below the editor, a status bar shows "View Problem" and "No quick fixes available". To the right of the editor, a message box displays the error: "SyntaxError: '(' was never closed".

**Kjøretidsfeil** lager feilmeldinger og programmet avbrytes. Det kan vi forhindre med **unntakshåndtering**. Kjørtidsfeil kalles **unntak** (*exceptions*), og de kan vi fange opp i en **try**-setning. Hvis det oppstår en feilmelding, vil koden i **try**-blokken avbrytes og koden i **except**-blokken kjøres. Hvis det ikke oppstår en feilmelding, vil koden i **try**-blokken kjøres normalt, og programmet vil hoppe over **except**-blokken. Vi kan ha flere **except**-blokker etter hverandre, og **try**-setningen kan også inneholde en **else**- og **finally**-blokk. Vi har nå en ny mulighet til å løse problemet med å gjøre om teksten til tall dersom teksten ikke består av bare siffer, `isnumeric()`.

```
tekst_inn = input('Skriv inn et heltall: ')
try:
    tall = int(tekst_inn)
    print('Du skrev inn ' + str(tall))
except ValueError:
    print(tekst_inn + ' er ikke et heltall.')
```

Det finnes mange typer unntak vi kan håndtere.

```
print(10/0)
```

ZeroDivisionError: division by zero

Vi kan fange opp alle unntak i **except**-blokken og skrive ut hvilken type unntak det er

```
try:
    fil = open('eksister.ikke')
except Exception as unntak:
    print('Type unntak: ' + unntak.__class__.__name__)
    print(unntak)
```

```
Type unntak: FileNotFoundError
[Errno 2] No such file or directory: 'eksister.ikke'
```

Imidlertid ønsker vi ofte å spare brukeren for mange av feilmeldingene.

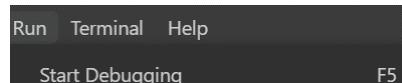
### Debugging og Logiske Feil

**Logiske feil** er vanligvis mer vrient å finne og rette. Vi kan finne feilene ved å teste programmet og rette dem ved å bruke et feilsøkingsprogram (*debugger*), se [KM06](#). VS Code hjelper oss med begge deler. La oss først se hvordan *debugger-en* virker.

Vi kan klikke til venstre for linjenummeret i koden for å sette et stoppunkt (*breakpoint*) som vises med en rød prikk.

```
1  for i in range(1,1001):
• 2      dobbelt = 2*i
```

Deretter kan vi starte *debugger-en* med **Run**, **Start Debugging** fra menylinja eller trykke **F5**.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Programmet vil stoppe der vi har satt et stoppunkt. Nå kan vi inspirere verdiene til variablene og navigere oss framover i programmet steg for steg. Det går også an å legge inn betingelser for når programmet skal stoppe. Da høyre-klikker vi på den røde prikken og velger **Edit Breakpoint...**. I eksempelet nedenfor er betingelsen `i==750`.

A screenshot of a Python debugger interface. On the left, there's a tree view under 'VARIABLES' with 'Locals' expanded, showing variables like 'special variables', 'dobbelts' (value 1498), and 'i' (value 750). In the center, there's a code editor window with the file 'b\_1\_7\_3.py' open, containing a for loop that calculates 'dobbelts' for each value from 1 to 1000. A red dot at the start of the second iteration indicates a breakpoint has been set at the condition 'i==750'. On the right, there's a toolbar with various icons.

### Testprogrammer og Assert-setninger

Før vi søker etter feil må vi vite hva vi leter etter. Vi skal lære å skrive testprogrammer som sjekker at koden vår fungerer som den skal og sier fra om den ikke gjør det. Det er mye mer effektivt enn å utføre en mengde manuelle tester for hver gang vi gjør endringer i koden. Nå skal vi nøye oss med å teste en funksjon med noen forskjellige verdier ved bruk av `assert` som er et [keyword](#) i Python.

```
1 def aldersgruppe(alder):  
2     if alder < 17: gruppe = 'barn'  
3     elif alder < 67: gruppe = 'voksen'  
4     else: gruppe = 'pensjonist'  
5     return(gruppe)  
6  
7 assert aldersgruppe(10)=='barn'  
8 assert aldersgruppe(15)=='barn'  
9 assert aldersgruppe(16)=='voksen'  
10 assert aldersgruppe(50)=='voksen'  
11 assert aldersgruppe(66)=='voksen'  
12 assert aldersgruppe(67)=='pensjonist'  
13 assert aldersgruppe(80)=='pensjonist'
```

Hvis betingelsene er riktige, skjer ingenting.

Hvis en betingelse er feil, lages et unntak (`AssertionError`) og programmet avbrytes.

```
\b_1_7_2.py", line 9, in <module>  
    assert aldersgruppe(16)=='voksen'  
AssertionError
```

Dersom vi starter *debugger-en* (med F5) i stedet for å kjøre programmet på normal måte (med Ctrl+F5), stopper utførelsen der hvor betingelsen feiler, og vi kan sjekke innholdet i variablene.

A screenshot of a Python debugger interface. The code is the same as above, but the line `assert aldersgruppe(16)=='voksen'` is highlighted with a red arrow pointing to it. A modal window titled 'Exception has occurred: AssertionError' is displayed, showing the error message 'exception: no description' and the stack trace: 'File "C:\Users\olali\OneDrive - Viken fylkeskommune\Dokumenter\\_IT-2\progs\b\_1\_7\_2.py", line 9, in <module> assert aldersgruppe(16)=='voksen'".

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

Det er nesten alltid feil i programmer, noe som forklarer at omtrent halvparten av tiden til å utvikle dem brukes til testing. Syntaksfeil gir kompileringsfeil og er typiske skrife feil. De vises med en rød bølget linje under ordet i VS Code, og bør rettes opp før vi prøver å kjøre programmet. Kjøretidsfeil kan være vanskelig å unngå, spesielt når vi leser inn data. Disse kan vi fange opp med unntakshåndtering (`try`- og `except`-blokker). Vi kan også ha logiske feil, som oppstår når programmet kjører uten feil, men gir galt resultat. Disse kan vi finne med testing og rette opp ved å studere koden eller bruke debuggeren. Vi kan teste om koden gir det forventede resultatet med `assert`-setninger, som vi senere skal bruke i testprogrammene våre.

### Øvingsoppgaver

#### 1.7.1

Finn og rett feilene i de vedlagte filene `p_1_7_1.py`. Dokumenter hva som var feil.

#### 1.7.2

Lag et program til å la brukeren skrive et tall mellom 1 og 7 for å velge en av regnbuens farger (rød, oransje, gul, grønn, blå, indigoblå og lilla). Programmet skal skrive ut "Flott! Du valgte <farge>." Bruk unntakshåndtering slik at programmet ikke avbrytes uansett hva brukeren skriver inn.

#### 1.7.3

Ta for deg oppgave 1.3.2 som handler om å oversette tall fra titallsystemet til binære tall. Løs denne oppgaven ved å benytte unntakshåndtering (med en `try`-blokk) i stedet for en `if`-setning med `isnumeric()`. Utforsk også muligheten for å benytte en `else`- eller `finally`-blokk i denne sammenhengen. Programmet skal altså fortsette å kjøre selv om brukeren taster inn noe annet enn et heltall.

#### 1.7.4

Ta for deg oppgave 1.4.5 om Fibonaccitallene under 1000. Sett et stoppunkt (breakpoint) i løkken som stopper ved det første Fibonaccitallet over 500. Ta et skjermbilde av *debugger-en* som viser innholdet i variablene til de to foregående Fibonaccitallene.

## 1.8 Modellering

### Bolkens innhold

Modeller i systemutvikling .....	43
Systemutvikling .....	43
Metoder for systemutvikling .....	44
Objektorientert modellering og programmering.....	44
Arv i objektorientert programmering .....	45
Bruk av UML i objektorientert modellering .....	45
Ta med minst dette.....	46
Øvingsoppgaver.....	47

### Kompetansemål:

- anvende objektorientert modellering til å beskrive et programs struktur.

I denne bolken vil vi gi en innføring i objektorientert modellering og hvordan det er en integrert del av systemutviklingsprosessen. Vi vil også sammenligne objektorientert programmering med andre programmeringsmetoder som er fundamentalt forskjellige. Vi vil utforske begrepet arv, som er sentralt innenfor objektorientert programmering, og presentere verktøyet PlantUML, et praktisk dataprogram som brukes til å lage spesifikke diagrammer og modeller.

### Modeller i systemutvikling

En **modell** er en representasjon av noe, og betegnelsen har ulike betydninger i forskjellige sammenhenger. For eksempel kan det referere til en nøyaktig [miniatyrversjon av et skip](#) i redusert skala eller en [tredimensjonal "walk-through" modell av en leilighetsblokk](#). En modell av et **dataprogram** er mer abstrakt og fungerer som en skjematisering. Modellering av et dataprogram innebærer å lage diagrammer som beskriver komponentene i programmet, deres innhold, samspill og forbindelser. Modellering av et program er vanligvis en aktivitet som utføres før vi begynner å skrive selve koden.

### Systemutvikling

**Datasystemer** er systemer for behandling, lagring og overføring av data, og de består av dataprogrammer. Prosessen med å utvikle datasystemer kalles **systemutvikling**. Systemutvikling innebærer flere faser eller aktiviteter, der programmering er bare én av dem.

# Smidig IT-2

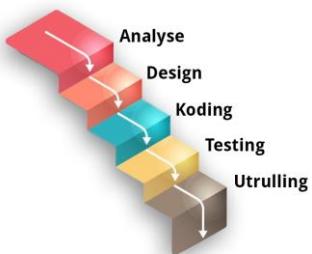
## for eksklusiv bruk ved <navn på skole>

Her er en vanlig inndeling av systemutviklingsprosessen i faser.

- **Analyse** : Identifisere **hva** som må gjøres
- **Design** : Bestemme **hvordan** det skal gjøres
- **Koding** : Skrive selve programmet
- **Testing** : Sørge for at programmet fungerer som forventet og rette eventuelle feil.
- **Utrulling** : Sette systemet i drift, **installasjon**, opplæring, vedlikehold, osv.

En betydelig del av modelleringen skjer i designfasen. Det kan være overraskende for mange å oppdage at bare 10-20% av tiden går med til selve programmeringen. Testing tar vanligvis lengst tid, kanskje rundt 50%, mens planleggingen (analyse og design) tar omtrent 25%. Det er vanlig at et systemutviklingsprosjekt involverer 5-10 systemutviklere og strekker seg over flere måneder. I planleggingsfasen (analyse og design) er diagrammer en effektiv måte å kommunisere ideer.

### Fossefallsmetoden



### Metoder for systemutvikling

Det finnes ulike tilnæringer for å gjennomføre prosjektfasene. Med den tradisjonelle [fossefallsmetoden](#) fullfører vi hver fase før vi går videre til neste. Dette kan skape problemer dersom kravene endrer seg underveis eller hvis feil oppstår sent i prosessen. Med [smidige metoder](#) gjentar vi aktivitetene flere ganger, gjerne med faste intervaller, for eksempel hver 4. uke. I løpet av denne perioden analyserer, designer, koder og tester vi ulike deler av

### Smidige metoder



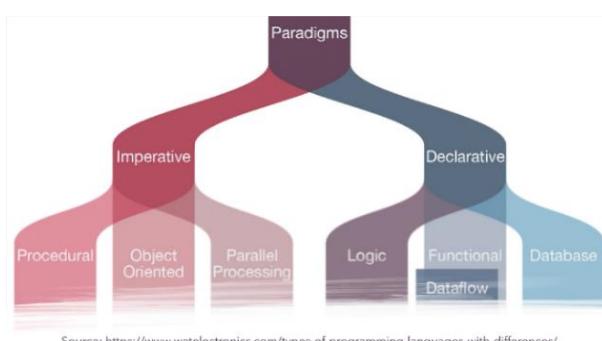
programmet. Selv om prosjektet strekker seg over ett helt år, vil vi hver måned ha stadig mer kode som er ferdig testet og fungerer. Prosjektstyring er viktig uansett hvilken tilnærming vi velger, da det ikke er uvanlig at IT-prosjekter mislykkes, noe som kan bekreftes gjennom et enkelt [søk](#) på internett.

### Objektorientert modellering og programmering

Vår modellering skal være objektorientert som vil si at vi skal bruke symbolene og språknotasjonene til *Unified Modeling Language (UML)*, [se de to siste avsnittene](#). Begrepene OOM, OOA, OOD og OOP står for henholdsvis Objekt-Orientert- Modellering, Analyse, Design og Programmering og er knyttet nært sammen. Målet med OOM, OOA og OOD er å hjelpe med å lage eller beskrive et objektorientert program. Objektorientert programmering er en av flere grunnleggende måter å programmere på, kjent som **programmeringsparadigmer**.

I **deklarativ** programmering forteller vi hva, men ikke hvordan. SQL og HTML er eksempler på dette. Vi skriver en SELECT-spørring og ber om poster som tilfredsstiller bestemte kriterier, sortert i en bestemt rekkefølge, men ikke hvordan det skal gjøres. Det tar SQL-tjeneren seg av.

Vi skriver en <IMG>-tagg for å plassere et bilde i en bestemt størrelse, i en bestemt



Source: <https://www.waterelectronics.com/types-of-programming-languages-with-differences/>

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

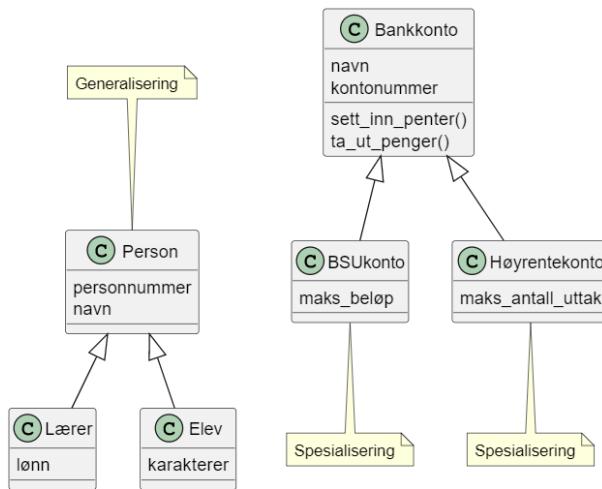
posisjon på en webside, men ikke hvordan det skal gjøres. Det tar nettleseren seg av. I **imperativ** programmering forteller vi både hva og hvordan datamaskinen skal utføre oppgaven. [finn maks](#)-funksjonen i 1.1 Grunnleggende programmering er et eksempel på **prosedyreorientert** programmering. [Person](#)-klassen i 1.2 Datatyper er et eksempel på **objektorientert** programmering. Både prosedyreorientert- og objektorientert programmering tilhører det imperative programmeringsparadigmet. Det finnes flere, se [Programming paradigm](#) (Wikipedia). Forskjellige programmeringsspråk støtter ett eller flere programmeringsparadigmer. For eksempel støtter Python imperativ, funksjonell, prosedyreorientert og objektorientert programmering.

### Arv i objektorientert programmering

Arv er et sentralt begrep i objektorientert programmering. En klasse kan arve egenskapene (variabler) og metoder (funksjoner) fra en annen klasse. Den underordnede subklassen (*child class*) kan bruke koden som er definert i den overordnede superklassen (*parent class*). Arv sparer tid og krefter, da vi unngår å skrive koden på nytt, samtidig som det gir en tydelig struktur som er lett å forstå.

Når vi har flere klasser som deler felles egenskaper og metoder, kan vi samle dem i en superklasse. Dette kalles **generalisering**. La os si vi har en Elev klasse (karakterer, osv.) og en annen Lærer klasse (lønn, osv.) Begge klassene inneholder navn og personnummer. Så vi oppretter superklassen Person for det som er felles. Vi kan også gå den andre veien. Det kalles **spesialisering**.

La oss si vi har en klasse Bankkonto som inneholder navn og kontonummer og metoder for å ta ut og sette inn penger. Så trenger vi flere og forskjellige egenskaper til Høyrentekonto og BSU-konto (Boligsparing for ungdom) og oppretter subklasser for disse. Arv er vårt andre eksempel på å generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode ([KM07](#)). Det første var funksjoner i seg selv. I Python angir vi arv med å skrive navnet til superklassen inni parenteser i definisjonen til subklassen: [class subklasse \(superklasse\)](#): Det skal vi gjøre i runde 2.



### Bruk av UML i objektorientert modellering

Når vi skal anvende objektorientert modellering til å beskrive et programs struktur, skal vi benytte **UML** (Unified Modeling Language). UML er ikke et programmeringsspråk, eller et språk vi snakker, men et modelleringsspråk, et system av grafiske framstillinger, et verktøy for å analysere og designe programvare, spesielt objektorienterte informasjonssystemer. UML er et visuelt språk med syntaks og semantikk, en [spesifikasjon](#) (på 796 sider) med regler for hvordan vi skal lage (14 standardiserte) diagrammer. Tenk "et bilde er sier mer enn tusen ord", fordelene med å snakke samme språk og "lage strukturen før koden for å unngå fallgruver". Det er vanligst å benytte UML som en skisse til å diskutere alternative løsninger med andre utviklere, men det benyttes også som dokumentasjon for å hjelpe oss til å forstå koden (f. eks når vi blir bedt om å endre på et program noen andre har laget).

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

UML diagrammer kan vi tegne for hånd, i vanlige tegneprogram eller vi kan bruke ett av mange programmer som er spesielt utviklet for å lage UML diagrammer. Vi skal bruke [PlantUML](#), som kan installeres som en utvidelse i VS Code. For at det skal fungere, må vi også installere **Java** og **Graphviz**. Vi kan klare oss uten Graphviz, da det kan være litt vrikt å installere på Mac. I stedet kan vi bruke [Smetana](#), som er innebygd i PlantUML, og angis med

`!pragma layout smetana` i koden. Lenkene ligger på siden til PlantUML siden i VS Code. Se disse tre videoene for å komme i gang med PlantUML:

1. [Using PlantUML in VSCode](#) (6 min)
2. [PlantUML - beautiful quick diagrams to explain your models](#) (16 min)
3. [PlantUML with Domain Models \(class diagrams\)](#) (10 min)

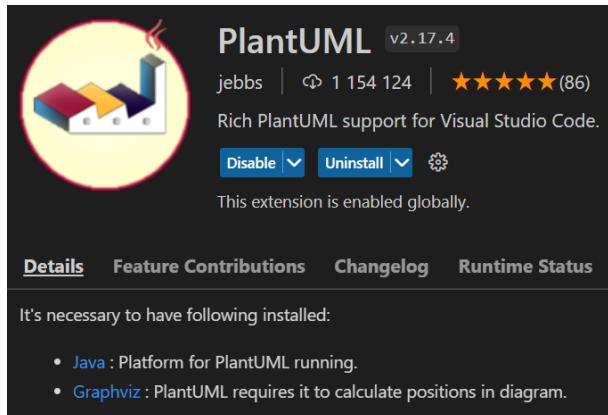
Vi finner mer informasjon på PlantUML sine [websider](#), og her er en [instruksjonsbok](#) som kan komme til nytte.

#### Ta med minst dette

Denne bolken har hovedsakelig vært teoretisk, bortsett fra PlantUML, som vi har brukt til å lage diagrammer. For å si det enkelt, kalles dataprogrammer av et litt større omfang for datasystemer, og arbeidet med å lage dem heter systemutvikling. Det deles gjerne inn i ulike aktiviteter som analyse, design, koding, testing og utrulling. Framgangsmåten kan variere, fra fossefallsmetoden hvor vi gjør oss ferdig med hver aktivitet før vi begynner på den neste, til smidige metoder, hvor alle aktivitetene utføres i flere runder, og vi bygger litt mer på hver aktivitet i hver runde.

Modellering av et dataprogram innebærer å lage diagrammer som beskriver komponentene i programmet, deres innhold, samspill og forbindelser. Dette arbeidet utføres i designfasen, før vi begynner å skrive selve koden. Det hjelper oss med å sikre at programmet blir riktig første gangen vi skriver koden, for å unngå tidkrevende og kostbare omgjøringer.

Når en klasse er underlagt en annen klasse i objektorientert modellering og programmering, arver den underlagte subklassen metoder og egenskaper fra den overordnede superklassen. Når flere subklasser er underlagt samme superklasse, oppnår vi gjenbruk av kode. Modellene (diagrammene) lager vi med PlantUML, en utvidelse i VS Code, som er enkel å bruke.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Øvingsoppgaver

#### 1.8.1

##### Installering

1. Installer PlantUML som en utvidelse i VS Code.
2. Installer [Java](#).
3. Installer [Graphviz](#). For Mac-brukere: Det kan være enklere å bruke Smetana som er innebygd i PlantUML istedenfor å installere Graphviz.

#### 1.8.2

Se på følgende videoene og svar på spørsmålene under:

4. [Using PlantUML in VSCode](#) (6 min)
  5. [PlantUML - beautiful quick diagrams to explain your models](#) (16 min)
  6. [PlantUML with Domain Models \(class diagrams\)](#) (10 min)
- a) Hvordan forhåndsviser du et diagram?
  - b) Hvordan lager du en bildefil av diagrammet?
  - c) Hvordan skiller PlantUML mellom egenskaper og metoder i klassediagrammer?
  - d) Hvordan viser vi at en klasse er en subklasse av en annen klasse?
- (Gå til <https://plantuml.com/>, velg *Class diagram* og søk etter (Ctrl-F) *extension*)

#### 1.8.3

Lag diagrammet som vises nedfor, og legg til 'Laget av <ditt navn>' som bunntekst.

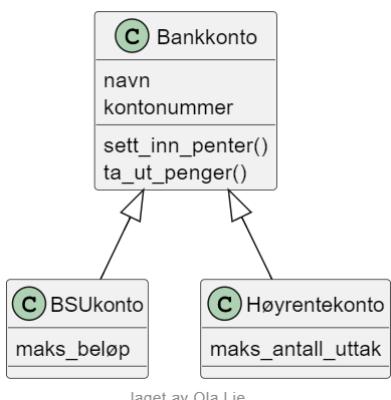
(Åpne [PlantUML Language Reference Guide \(depu.js.org\)](#) og søk etter *Title Legend* ).

Lever diagrammet som

- a) Bildefil i valgfritt format egnet for internett
- b) PDF-dokument

(Bruk Widows-tasten + Shift + S for å kopiere, lim inn i Word og lagre som PDF)

Arv



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.9 Reelle datasett med CSV

#### Bolkens innhold

Åpne data.....	48
HTTP requests.....	49
Anonyme funksjoner.....	50
Valgoperatøren .....	51
csv.....	52
pandas .....	52
seaborn.....	52
Ta med minst dette.....	53
Øvingsoppgaver.....	54

#### Kompetanse mål:

- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett.

#### Åpne data

Åpne data er datasett som vi kan bruke gratis, fordi de ikke er begrenset av opphavsrett eller lisenser. Offentlige og andre virksomheter tilbyr [åpne data](#). Det er ingen selvfølge at data er gratis, for det ligger mye penger i dem, se [The world's most valuable resource is no longer oil, but data](#) (The Economist). På nettet finnes mye åpne data som vi kan bruke, og <https://data.norge.no/> (Felles datakatalog) er et greit utgangspunkt. Vi kan laste ned filer eller la programmet vårt hente informasjon via API (Application Programming Interface).

The screenshot shows the Felles Datakatalog homepage with a search bar containing 'Eksempel: kollektivtransport'. Below the search bar are five cards representing different types of data assets:

- Vis kun datasett (Icon: stack of documents)
- Vis kun api (Icon: gear)
- Vis kun begrep (Icon: book)
- Vis kun informasjonsmodell (Icon: tree)
- Vis kun tjeneste og hendelse (Icon: document with yellow 'BETA' banner)

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

I denne bolken skal vi bruke [Statens satser utland 2019](#) som viser statens satser for dekning av utgifter til reise og kost utenfor Norge. La oss si vi ønsker å finne de fem dyreste og de fem billigste stedene, samt gi en grafisk presentasjon av dataene.

Filtypen er **.csv**. Det finnes også [informasjon om datasettet](#). Når vi jobber med store datasett, er det vanlig å foreta en foreløpig undersøkelse av datasettet, for å få et førsteinntrykk og bli bedre kjent med innholdet. Vi kan se på filen (2019.csv) i VS Code. Vi ser at det er én overskriftsrad og at skilletegnet er semikolon (;), ikke komma (.). Flere rader mangler by og administrativt\_satt, og en rad har ikke kost.

```
2019.csv
1 verdensdel;land;by;kost;administrativt_satt
2 Afrika;Øvrige områder;;560;
3 Afrika;Algerie;;500;
40 Amerika;Brasil;;560;
41 Amerika;Brasil;Rio de Janeiro;620;
119 Europa;Danmark;;1220;
120 Europa;Danmark;København;1220;
121 Europa;Færøyene;;1220;Danmark
122 Europa;Grønland;;1220;Danmark
113 Europa;Øvrige områder;;
```

Det er vanlig å "vaske" dataene før vi bruker dem. Vi må blant annet bestemme hva vi gjør med rader som har noen verdier, men mangler andre. I vårt tilfelle dropper vi linje 1 (overskriftsraden) og linje113 (uten kost). Begge linjene vil feile hvis vi prøver å gjøre om **kost** til **int** i en rad når verdien er en tom tekst, ", eller inneholder noe annet enn siffer.

#### HTTP requests

Vi kan lese inn filen slik vi har lært tidligere, eller vi kan laste den ned i programmet med [http requests \(w3schools\)](#).

```
import requests # Husk: pip install requests
url      = 'https://hotell.difi.no/download/statens-satser/utland/2019'
r       = requests.get(url)  # returnerer et Response objekt
```

Dersom noe går galt med forespørselen, får vi feil, for eksempel *ConnectionError*, som vi kan fange opp med unntak. Går alt bra, returner [get](#)-metoden et [Response](#)-objekt ([W3Schools](#)) som vi kan utforske med vår variabel **r**.

```
print(r)                      # HTTP 200 OK success status response code
<Response [200]>
print(r.content[:60])          # vis første 60 bytes
b'verdensdel;land;by;kost;administrativt_satt\r\nAfrika;\xc3\x98vrige
print(r.text[:74])              # vis først 74 tegn
verdensdel;land;by;kost;administrativt_satt
Afrika;Øvrige områder;;560;
```

**content**-egenskapen viser innholdet som bytes, og vi kan se **\r\n** mellom linjene og 'Ø' kodet som **\xc3\x98**. **text**-egenskapen viser innholdet som tekst. Dersom **r.encoding** ikke er satt, tipper programmet på beste tegnkoding (utf-8) for oss.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Verdiene i teksten kan vi legge inn i en tabell. Så kan vi utføre en datavask, konvertere tekst til tall og sortere tabellen. Vi begynner med å lage en liste med linjene.

```
linjer = r.text.split('\r\n') # lag en liste med linjene
print(linjer[:2])           # vis 2 først elementer i listen
['verdensdel;land;by;kost;administrativt_satt', 'Afrika;Øvrige områder;;560;']
```

Så lager vi en liste av hver linje og vasker bort rader som ikke har tall (som tekst) i i **kost**-feltet samtidig som vi konverterer tekst til tall. Vi får da en todimensjonal tabell (lister i liste).

```
tabell = []

for linje in linjer:
    liste = linje.split(';') # lag en liste for hver linje

    # Overskriftsrad og rader uten kost tas ikke med (1 og 113)
    if liste[3].isnumeric():
        liste[3] = int(liste[3])
        tabell.append(liste)

print(tabell [0:2])          # vis to første rader i tabellen
[['Afrika', 'Øvrige områder', '', 560, ''], ['Afrika', 'Algerie', '', 500, '']]
```

Før vi sorterer tabellen skal vi se nærmere på anonyme funksjoner.

#### Anonyme funksjoner

Anonyme funksjoner ([Lambda functions](#)) er funksjoner uten navn som ofte blir brukt som argumenter nå vi kaller andre funksjoner. Husk at en funksjons **parametere** er navn som er oppført i funksjonens definisjon. **Argumentene** er de virkelige verdiene som sendes til funksjonen. Vi har tidligere sett at vi kan sortere lister med tall fra lavest til høyest eller tekst alfabetisk.

```
farger = ['oransje', 'gul', 'grønn']
farger.sort()
print(farger)
['grønn', 'gul', 'oransje']
```

Hvis vi ønsker å sortere tabellen etter andre kriterier, kan vi lage en funksjon for dette.

```
def ordlengde(ord):
    return len(ord)

print(ordlengde('Hvit'))
4
farger.sort(key = ordlengde)
print(farger)
['gul', 'grønn', 'oransje']
```

I stedet for å lage funksjonen `ordlengde()`, kan vi sende den anonyme (`lambda`) funksjonen som argument til `sort`-metoden, og vi slipper å skrive like mye.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
farger.sort(reverse=True, key = lambda x :len(x))
print(farger)
['oransje', 'grønn', 'gul']
```

Formatet er **lambda parametere : uttrykk**. Vi kan nå sortere tabellen for statens satser utland etter verdiene i **kost**-feltet, kolonne 4, indeks 3.

```
sortert_tabell = sorted(tabell, reverse=True, key=lambda x: x[3])
for i in range(5):print(sortert_tabell[i])
['Amerika', 'Venezuela', '', 1600, '']
['Europa', 'Liechtenstein', '', 1240, '']
['Europa', 'Sveits', 'Geneve', 1240, '']
['Europa', 'Sveits', 'Zürich', 1240, '']
['Europa', 'Danmark', '', 1220, '']
```

Før vi formaterer utskriften, skal vi se på valgoperatøren.

### Valgoperatøren

Hvis vi ønsker at en variabel skal få én av to mulige verdier, kan vi bruke valgoperatøren ([Conditional Expressions/ternary operator](#)). Nedenfor vises et eksempel som skriver ut antall timer, men hvis antallet er 1, heter det jo time i entall.

```
for i in range(4):
    tidsenhet = 'timer' if i!=1 else 'time'
    print(f'{i} {tidsenhet}')
print()
for i in range(4):
    print(f'{i} {"timer" if i!=1 else "time"}')
```

I uttrykket **x if B else y** blir først betingen B evaluert. Hvis B er sann, blir x evaluert og verdien 0 timer returnert, ellers blir y evaluert og verdien 1 time 2 timer 3 timer returnert.

I tabellen for statens satser utland ønsker vi å skrive **by**, **land** hvis by ikke er en tom tekst, ellers skriver vi bare **land**.

```
print("\n*** De 5 dyreste stedene ***")
print("Sted                      Kost")
print("*****")
for rad in sortert_tabell[:5]:
    sted = f'{rad[2]}, {rad[1]}' if rad[2] != "" else rad[1]
    print(f'{sted:25}{rad[3]:7}')
print()
*** De 5 dyreste stedene ***
Sted                      Kost
-----
Venezuela                  1600
Liechtenstein               1240
Geneve, Sveits              1240
Zürich, Sveits              1240
Danmark                     1220
```

Vi venter med de 5 billigste stedene til oppgavene. Vi kunne sluppet unna med mindre programmering om vi hadde brukt csv-modulen (.py-fil).

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### CSV

[csv](#)-modulen kan lese og skrive csv-filer med tabelldata, så det gir oss litt hjelp på veien. Videre bruker vi en mer kompakt kode for å bygge opp tabellen, og [itemgetter](#) for å hente verdier fra **kost**-feltet (kolonnen) når vi sorterer. [:5] kalles [slicing](#) (snitte opp) og henter de fem første elementene. Stjerneoperatøren \*<sup>8</sup> pakker ut den sorterte listen og `sep='\\n'` skiller elementene med linjeskift. Det siste er kun gjort for å få hvert element på en ny linje. Vi kunne godt ha formatert utskriften som ovenfor.

```
import csv
from operator import itemgetter
csv_reader = csv.reader(requests.get(url).text.splitlines(), delimiter=';')
tabell = [x for x in list(csv_reader) if x[3].isnumeric()]
for rad in tabell: rad[3]=int(rad[3])
print(*sorted(tabell, reverse=True, key=itemgetter(3))[:5], sep='\\n')
```

```
['Amerika', 'Venezuela', '', 1600, '']
['Europa', 'Liechtenstein', '', 1240, '']
['Europa', 'Sveits', 'Geneve', 1240, '']
['Europa', 'Sveits', 'Zürich', 1240, '']
['Europa', 'Danmark', '', 1220, '']
```

### pandas

Enklest er det å bruke pakken [pandas](#)<sup>9</sup>. En pakke er en samling moduler (.py-filer). pandas er godt egnet til arbeide med todimensjonale tabeller og mye annet. Med pandas kan vi løse hele oppgaven med noen få linjer.

```
import pandas as pd # Husk: pip install pandas
df = pd.read_csv(url, sep=';')
print(df.sort_values(by='kost', ascending=False).head(5).fillna(''))
```

	verdensdel	land	by	kost	administrativt_satt
62	Amerika	Venezuela		1600.0	
136	Europa	Liechtenstein		1240.0	
167	Europa	Sveits	Geneve	1240.0	
168	Europa	Sveits	Zürich	1240.0	
118	Europa	Danmark	København	1220.0	

`read_csv`-metoden returnerer et [DataFrame](#)-objekt med rader og kolonner som kan sorteres direkte på kolonnenavn fra overskriftsraden. `head(5)` vil si at vi ønsker kun de første fem radene, og `fillna('')` bytter ut [NaN](#) (*not a number*) med en tom tekst. I en [DataFrame](#) betyr [NaN](#) at det mangler data.

### seaborn

[seaborn](#) er et bibliotek (pakke) til grafiske fremstillinger. [seaborn](#) bygger på [matplotlib](#), men er enklere å bruke og er tett integrert med pandas.

Et boksplott er et diagram som viser fordelingen av dataene og gir indikasjoner på symmetri i dem. Diagrammet gir et oversiktlig sammendrag så vi kan raskt og effektivt oppsummere og

<sup>8</sup> Se [stjerneoperatøren](#) i 2.2 Tupler og ordbøker

<sup>9</sup> Vi skal lære mer om pandas og seaborn i runde 3.

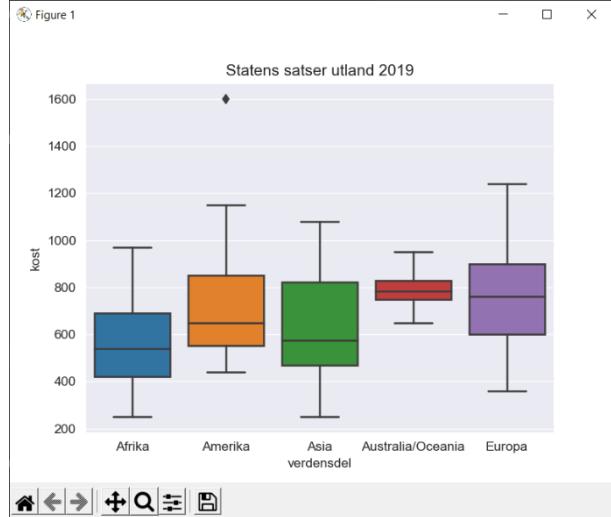
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

sammenligne forskjellige sett med data.

```
import seaborn as sns          # Husk: pip install seaborn
import matplotlib.pyplot as plt # Husk: pip install matplotlib
sns.set_style('darkgrid')
sns.boxplot(data=df,x='verdensdel',y='kost') \
    .set(title='Statens satser utland 2019')
plt.show()
```

Streken i midten av boksene viser medianen. En boks strekker seg fra nedre til øvre kvartil, så halvparten av alle verdiene ligger innenfor boksen. Enden på strekene (værhårene/whiskers) viser minimum- og maksimum-verdiene, men ikke lengre unna boksen enn 1.5 ganger kvartilbredden (boksenes høyde). Verdier på utsiden av værhårene kalles uteliggere og vises med diamant-symboler. Vi må bestemme oss for hva vi vil gjøre med uteliggere, for verdiene er så avvikende at de kan være feil. Det blir en vurderingssak å tolke hvorvidt uteliggere bør være med i analysen eller ikke.



### Ta med minst dette

Åpne data er gratis og ikke begrenset av opphavsrett eller lisenser. Vi kan hente data (lese filer) fra Internett med bruk av biblioteket `requests` og funksjonen `get`. Dersom det er en CSV-fil, må vi bygge opp tabellen selv. Da er det enklere å bruke biblioteket `pandas` som har funksjonen `reader`. Den returnerer et `reader`-objekt som er en *iterator* vi kan bruke i en `for`-løkke. Hver gjennomgang returnerer en linje som en liste med tekststrenger. Enda lettere er det å bruke `pandas`-biblioteket og `read_csv()` som returnerer innholdet i filen som en `DataFrame`. En `DataFrame` er en tabell med rader og kolonner. Med `pandas` er det enkelt å gruppere, sortere og utføre andre operasjoner på tabellen, noe vi skal lære mer om senere og bruke mye. Vi kan bruke `csv.reader()` direkte på en lokal CSV-fil, men når vi bruker den med en URL, som i `csv.reader(requests.get(<url>).text.splitlines())`, hentes teksten som en enkelt tekststreng som vi må splitte selv. `pandas.read_csv()`, derimot, tillater både en lokal filsti eller en URL som argument, og håndterer resten automatisk.

`Seaborn` er et bibliotek tilpasset `pandas` for å lage diagrammer. Valgoperatøren er ikke noe annet enn å skrive en `if`-blokk på en mer kompakt måte. Når vi bruker funksjoner som krever en annen funksjon som argument, kan vi bruke en anonym `lambda`-funksjon til å definere denne funksjonen direkte i funksjonskallet.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Øvingsoppgaver

1.9.1

Statistisk sentralbyrå tilbyr mye åpne data. Utforsk [www.ssb.no](http://www.ssb.no) for å gjøre deg litt kjent med websidene. Foruten ferdige tabeller kan vi velge ut data selv og laste dem ned som regneark eller med API-spørrenger, se eksempelvis [06913: Endringer i kommuner, fylker og hele landets befolkning \(K\) 1951 - 2022. Statistikkbanken \(ssb.no\)](#)

1.9.2

Statistisk sentralbyrå tilbyr også ferdige datasett i csv-format til de mest brukte tabellene. Utforsk hvilke datasett som er tilgjengelig på [API: Ferdige datasett \(ssb.no\)](#)

1.9.3

I denne oppgaven skal du analysere samlivsformer i Norge. Gifter flere eller færre seg nå enn tidligere? Lag et program som laster ned csv-filen fra

[Personer 18 år og over, etter samlivsform, i prosent. Hele landet, 2005 - siste år](#)

Forklar hvor du finner lenken til csv-filen: <https://data.ssb.no/api/v0/dataset/86813.csv>

Utforsk [seaborn.lineplot](#) og lag et linjediagram med år langs x-aksen. De ulike samlivsformene skal vises i prosent langs y-akse. Diagrammet skal ha tre linjer; en for hver av samlivsformene.

#### TIPS

Denne filen bruker ikke [utf-8](#), men [windows-1252](#), og vi bør fortelle om komma som desimaltegn, så vil slipper å gjøre det om senere:

```
df = pd.read_csv(url, sep=';', decimal=',', encoding='windows-1252')
```

Vi får tak i en kolonne gjennom kolonnenavnet som hentes fra overskriftsraden. Hvis denne teksten er svært lang, som i dette datasettet, kan vi endre på kolonnenavnet eller angi kolonnen ved indeks.

```
df.iloc[:, 4] der det samme som  
df.columns.values[4] = "prosent"  
df['prosent']
```

Hvis årstallene blir gjort om til desimaltall i grafen, er det flere måter å løse det på. Et alternativ er å gjøre årene, som er heltall, om til [Timestamp](#) objekter.

```
df["år"] = pd.to_datetime(df["år"].astype(str), format="%Y")
```

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 1.10 Informasjonsteknologi

### Bolkens innhold

Muligheter, utfordringer og konsekvenser.....	55
Etiske dilemmaer.....	57
Ta med minst dette.....	58
Øvingsoppgaver.....	59

### Kompetansemål:

- utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger
- drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn

### Muligheter, utfordringer og konsekvenser

Det får opplagt mindre konsekvenser når en [kasteknall](#)<sup>10</sup> eksploderer enn når en atombombe detonerer, for størrelsesordenen spiller en stor rolle. Så hvor stor er informasjonsteknologi? I 2023 omsatte Apple for 383 milliarder dollar eller 4 062 milliarder norske kroner<sup>11</sup>. Den norske stats inntekter dette året var til sammenligning 2 494 milliarder. I 2024 var 6 av de 10 største bedriftene i verden IT-selskaper, inkludert Microsoft, Apple, Alphabet (Google), Amazon, Nvidia og Meta (Facebook), verdt billioner av kroner ([Investopedia](#)). Det har ikke alltid vært slik. Tju år tidligere var bare to IT-selskaper på denne listen ([YouTube](#)).



I 2021 var informasjonsteknologi den 6. største industrisektoren i verden etter finansielle tjenester, byggenæringen, eiendomsmegling, E-handel og livs- og helseforsikring ([Yahoo](#)). Da regnes telekommunikasjon som en egen sektor som kom på 10. plass, og det er flere mobiltelefonabonnement enn mennesker på jorda ([Hindustan times](#)). IT står for omrent 10% av verdens samlede energiforbruk ([Wikipedia](#)), og det jobber over 10 ganger så mange mennesker i IT-bransjen som det er innbyggere i Norge ([statista](#)). Så når vi skal studere konsekvenser av og bruk av informasjonsteknologi bør vi være bevisst hvor stor rolle den inntar i samfunnet vårt. Det er vanskelig å forestille seg dagens samfunn uten IT. Hvor avhengige er vi egentlig av informasjonsteknologi? Hva skjer på flyplasser, togstasjoner og sykehus dersom datasystemene svikter? Hva skjer i butikkene når vi skal betale dersom banksystemene bryter sammen og stadig færre folk har kontanter? Hva hvis

<sup>10</sup> Se også [Kinaputt](#)

<sup>11</sup> Valutakurs pr 08.06.2024

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

kommunikasjonssystemene til politiet og forsvaret blir satt ut av spill? Hvor lenge kan vi være uten elektrisk strøm?



#### Et hav av muligheter

Ja, informasjonsteknologi har utvilsomt inntatt en stadig mer framtredende rolle i samfunnet. Fra å støtte utviklingen, for så å inngå i den, til å drive den framover. Til å begynne med ble informasjonsteknologi brukt til å hjelpe kontorfunksjonærer med arbeidsoppgaver innen økonomi, administrasjon og produksjonsplanlegging. Så robotiserte vi mye av produksjonen, og det ble færre

fabrikk- og lagerarbeidere. Etter hvert inngikk IT i mange eksisterende produkter som vaskemaskiner, støvsugere, telefonnettet, radioer, TV-er, stereoanlegg, kamera, klokker, mobiltelefoner, biler, osv. Produktene fikk både nye og bedre funksjoner ved bruk av IT. En rekke nye kommunikasjonsmuligheter (Skype, Teams, Zoom, mobilt internett, ZigBee, Z-Wave, ...) åpnet igjen veien for nye produkter som digital undervisning, telemedisin og smarte hjem. Det samme kan sies om droner og kunstig intelligens som har gitt oss [gjenbrukbare fyrverkeri](#), [flygende såmaskiner](#), [laserlukere](#), [talekontrollsystemer](#) (som Apples Siri, Google Assistant) og Amazons Alexa) og [selvkjørende biler](#) som trenger verken ratt eller pedaler. Det forventes at denne utviklingen vil fortsette de kommende årene, også innen virtuell og utvidet virkelighet, tingenes internett, 3D-utskift, biometri og andre spennende områder.

Så informasjonsteknologi skaper åpenbart nye muligheter. Facebook ble etablert i 2004, Amazon og Google på 1990-tallet. Disse selskapene er relativt unge sammenlignet med andre store, internasjonale selskaper fra olje- og bilindustrien som er over 100 år gamle. Den første mobilappen kom i 1997, og smarthusteknologi etter dette. Flere nordmenn har blitt milliardærer på informasjonsteknologi de siste 10 årene ([Innovatørene kommer!](#)).

Nyetableringer skaper arbeidsplasser. Sysselsetting er bra rent økonomisk for individ og samfunn. Mer penger til lønninger leder til et høyere forbruk som andre selskaper tjener på, som igjen leder til et høyere forbruk. Samfunnet mottar økte skatteinntekter og har mindre utgifter til arbeidsledighetstrygd. Vi får mindre fattigdom, høyere levealder og kanskje et bedre utdanningstilbud. Så sysselsetting har positive ringvirkninger. Dessverre fører høyt forbruk samtidig til mer forurensning, inflasjonspress og andre uønskede konsekvenser. Mye tyder likevel på at fordelene har overgått ulempene siden informasjonsteknologi har blitt så allment utbredt. Dessuten synes mange unge det er gøy å studere IT. IT-studentene blir fortalt at de er " fremtiden ", og selskapene står nærmest i kø for å sponse studentene med mangfoldige millioner kroner til sosiale aktiviteter. På UiO får 9 av 10 IT-studenter jobb før de er ferdige med utdanningen <sup>12</sup>. Foruten å bli drevet fram av økonomiske motiver, kommer mye av innovasjonen innen IT også fra kreativitet og gleden ved å finne ut av ting.

<sup>12</sup> Foredrag av IT-student ved UiO, Gyda Sæter, hos Askim vgs 12.09.2022

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Da utenlandske selskaper ville etablere datasentre i Norge, lokket kommunene med gratis tomter og billig strøm. I 2021 var det fremdeles slik at regjeringen ville lokke datasentre til Norge ([Regjeringen.no](#)). Det har ikke alltid gått bra, som i 2017 da "Ballangen ble lovet 3000 arbeidsplasser i bytte mot gigantisk tomt. Fire år senere havnet store summer i private lommer, og kommunen sto igjen med ei millionregning. Hva gikk egentlig galt?" ([NRK](#)). I 2021 var det igjen 5 arbeidsplasser (["Tre gladsaker og en tragedie"](#)).

Dessuten hadde det blitt mer søkelys på blokkjedeteknologien, som blir brukt til å lage kryptovaluta. Foruten å være et betalingsmiddel eller en investering, forbinder kryptovaluta også med kriminalitet, hvitvasking, svart økonomi, utpressing og svindel ([Et instrument for svindel](#)). Politikerne på Stortinget var uenige i hvorvidt datasentre som drev med kryptovaluta skulle få billigere strøm, og regjeringen snudde et tidligere vedtak og lot likevel datasentre som driver med kryptovaluta slippe full elavgift i 2021 ([E24](#)). I 2022 tok Sortland kommune alle midler i bruk for å unngå strømslukende kryptoanlegg ([NRK](#)). Er det noe mer? Det er det alltid. Hva med å få søvnforstyrrelser og livskvaliteten redusert på grunn av [støy fra datasentre](#)?



La **kryptovaluta** være vårt første eksempel på muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi. Samtidig skaper kryptovaluta et etisk dilemma for samfunnet: Hvis vi sier ja til disse arbeidsplassene og inntektene, er vi samtidig med på å støtte kriminalitet? Uansett, er det ikke bedre at datasentrene driftes i Norge på fornybar energi enn i områder med fossil kraftproduksjon? Så det er riktig at store deler av befolkningen skal betale høye strømpriser mens utenlandske selskaper får strømmen vår billig for å tjene enda mer penger?

#### **Etiske dilemmaer**

NDLA skriver "[Etikk](#) er refleksjoner over moralen. Det kan også defineres som læren om moral. I etikken er en opptatt av å begrunne hvorfor noe er rett eller galt og tenke systematisk gjennom våre oppfatninger av godt og ondt.", og om et [etisk dilemma](#): "I noen situasjoner er det vanskelig å avgjøre hvilken handling som er rett. Et dilemma er en situasjon der man har flere alternativer, og der ingen av alternativene byr på en god løsning på problemet."



Vi har sett ovenfor at samfunnet står overfor et etisk dilemma når det gjelder datasentre som lager kryptovaluta. På den ene siden er det "uheldig med en særnorsk regel som skaper usikkerhet og hindrer etablering av datasentre i Norge". På den andre siden ønsker ikke myndighetene å støtte

virksomhet som er til hjelp for kriminelle. Enkeltindivider kan også stå overfor etiske dilemma som en konsekvens av informasjonsteknologi. De senere år har det vært satt søkelys på



**Området er  
kameraovervåket**

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

unge som deler nakenbilder av seg selv. Press, ønske om popularitet og bedre selvtillit kan få dem til å gjøre det. Frykten for ryktespredning, hets, trakassering og skamfølelse taler imot ([Redd Barna](#)). En tredje type etisk dilemma oppstår når det som gagner samfunnet blir galt for individet. Et eksempel er overvåkning for å ivareta samfunnets behov for sikkerhet opp imot individets rett til personvern. Dette ble debattert mye i USA etter 9/11 da myndighetene vedtok loven [USA Patriot Act](#) som gjorde det lettere for regjeringen å spionere på vanlige amerikanere. Som oftest skjer overvåkningen ved bruk av og ved hjelp av informasjons-teknologi. Samtidig er [retten til privatliv](#) viktig i et demokratisk samfunn og er nedfelt i menneskerettighetskonvensjonen og i mange lands lover, som i vår egen grunnlov. At overvåking foregår, er de fleste klar over. Det kan dog bli en pinlig affære, som i 2013 da [Edward Snowden](#) avslørte at amerikanske myndigheter hadde avlyttet mobiltelefonen til den tyske forbundskansleren Angela Merkel.

Det er mange mulige etiske dilemmaer som kan oppstå som en konsekvens av informasjonsteknologi og vi vil drøfte noen av dem senere. Her er et utvalg å velge mellom

- personvern
- tilgangsrettigheter
- uoversiktlige ansvarsområder
- bruk av data fra tingenes internett
- informasjonsteknologiens effekt på miljøet
- alltid pålogget – skille mellom arbeidstid og fritid
- opphavsrett
- erstatningsansvar
- piratkopiering
- våpenteknologi
- bioinformatikk
- helseovervåkning
- feilinformasjon og deepfakes
- virtuelle organisasjoner
- bedriftshemmeligheter
- patenter
- kunstig intelligens
- datadeling
- cybersikkerhet
- masselagring av personlig informasjon
- autonom teknologi
- driftsutsetting

#### **Ta med minst dette**

Informasjonsteknologi har vokst til å bli en dominerende kraft i den globale økonomien, med store selskaper som Microsoft, Apple og Alphabet som frontfigurer. Samtidig som IT har gitt oss et hav av nye produkter og muligheter, som mange har blitt rike på, påvirker det også samfunn og individ i alt fra teknologiske fremskritt, økonomisk vekst, sysselsetting og skatteinntekter til energiforbruk og miljø, kritisk infrastruktur og personvern. Med de

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

mulighetene og konsekvensene dette medfører, fremkommer også etiske dilemmaer i kjølvannet av denne utviklingen. Etiske dilemmaer oppstår når vi står overfor valg hvor ulike moralske hensyn, som plikter, konsekvenser, og personlige holdninger, kommer i konflikt, og der ingen av valgene fremstår som entydig riktige. Det er fordeler og ulemper forbundet med hvert av valgene.

### Øvingsoppgaver

#### 1.10.1

Søk på nettet eller begynn med denne [nettsiden](#) for å finne fem av de største IT selskapene i Norge og svare på følgende spørsmål

- a) Hvor mye omsetter de for?
- b) Hvor stort overskudd har de?
- c) Hvor mange ansatte har de?
- d) Hvilke (hoved) produkter tilbyr de? Et produkt kan være varer eller tjenester.

#### 1.10.2

Regjeringen ønsker at Norge skal bli verdens mest bærekraftige datasenternasjon. Som vi har sett, kan dette medføre en rekke fordeler og ulemper. Drøft ett av de etiske dilemmaene som oppstår grunnet denne politikken. Når du drøfter dilemmaet du har valgt, bør du anvende disse modellene for etisk refleksjon, husk også at å drøfte betyr å se en sak fra forskjellige sider ta stilling.

Publisert under: Regjeringen Solberg  
Utgiver: Kommunal- og moderniseringssdepartementet

### Vil bli verdens mest bærekraftige datasenternasjon

Pressemelding | Dato: 11.08.2021

- Norge har et unikt utgangspunkt for å bli verdens mest attraktive datasenternasjon. Vi har overskudd på fornybar energi, lave strømpriser, god digital infrastruktur og et kjølig klima. Regjeringen styrker nå satsingen på en bærekraftig datasenterindustri. Det vil gi mange nye arbeidsplasser i distriktene og bidra til å utvikle nye digitale tjenester i hele landet, sier distrikts- og digitaliseringsminister Linda Hofstad Helleland.

- a) Konsekvensetikk, en modell som forsøker å belyse positive og negative konsekvenser av en handling. Deretter må man veie de positive og negative opp mot hverandre. Hvilken side veier tyngst? Hvorfor?
- b) Pliktetikk er en modell for refleksjon som spør etter hvilke plikter man har som menneske. Vi har plikt til å følge loven, men også til å følge uskrevne normer og kanskje også vår egen overbevisning? Hvilke lover og uformelle normer er i spill i dilemmaet, og hva forteller vår egen kritiske tenkning oss?
- c) Holdningsetikk, en modell som forsøker å svare på hva som er rett og galt ifølge våre verdier, våre holdninger og vår «hjertefølelse». Vi vil alle hjelpe mennesker, men gjør vi det ved å la dem få alt de peker på? Hvordan kan vi argumentere for eller imot å etablere datasentre utfra holdningene våre til naturen, menneskene, økonomien vår, kriminalitet og liknende?

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 1.11 Oppsummering

*utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger*

Først har vi sett på den stadig større plassen informasjonsteknologi tar i samfunnet. Mange er helt avhengige av informasjonsteknologi for å kunne utføre jobben sin. Videre inngår informasjonsteknologi i et mangfold av produkter, og nye varer og tjenester har blitt utviklet ved bruk av informasjonsteknologi. Samtidig som informasjonsteknologi har blitt så allment utbredt, har vi også blitt mer sårbar, og når vi løser et problem, oppstår et annet, [KM01](#).

*drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn*

Et etisk dilemma oppstår når vi har flere alternativer, og ingen av dem byr på en god løsning, så det blir vanskelig å avgjøre hva som er rett eller galt. Mange etiske dilemmaer oppstår i kjølvannet av informasjonsteknologi. Vi har drøftet problemstillinger på samfunnsnivå med datasentre som lager kryptovaluta, på individnivå med deling av nakenbilder og samfunnets behov for overvåkning som er en inngrisen i individets personvern, [KM02](#).

*utforske og vurdere alternative løsninger for design og implementering av et program*

Vi har løst enkelte oppgaver på flere forskjellige måter og har sett at ulik kode kan oppnå samme resultatet. I stedet for å kode alt selv har vi i tillegg brukt ferdige funksjoner som er innbygget i Python eller som vi har importert fra andre moduler, [KM07](#). Disse alternative løsningene har vi vurdert opp mot hverandre, [KM03](#). Det er enklere og går forttere å bruke ferdige funksjoner selv om vi lærer mer av å kode selv. Det siste vil sette oss bedre i stand til å utvikle gjenbrukbar programkode, [KM07](#).

*anvende objektorientert modellering til å beskrive et programs struktur*

Objektorientert modellering vil si å lage UML diagrammer som beskriver et programs struktur. Målet med modellene er å hjelpe oss med å lage og forstå objektorienterte programmer. Vi har sett at mye av modelleringen foregår i designfasen i systemutviklingsprosessen, og at det finnes andre måter å programmere på som ikke er objektorienterte. UML er ikke et vanlig språk, men et visuelt språk med regler for hvordan vi skal tegne spesielle diagrammer. Vi har installert PlantUML i VS Code som hjelper oss med dette, [KM04](#).

*utvikle objektorienterte programmer med klasser, objekter, metoder og arv*

Begrepene **klasser**, **objekter**, **egenskaper** og **metoder** er sentrale i objektorientert programering, og vi har sett at alt i Python er objekter. Videre har vi utforsket metodene til flere innebygde datatyper som **str**, **int**, **float** og **bool**. Datatyper er i virkeligheten klasser og variabler er referanser til objekter fra disse klassene. Vi har også fått en innføring i hvordan subklasser arver egenskaper og metoder til superklasser, [KM05](#).

Vurderingsksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

vurdere og bruke strategier for feilsøking og testing av programkode

Alle som programmerer gjør feil. Jo forttere vi finner og retter feilene, desto bedre. Vi har lært om ulike typer feil: Syntaksfeil, semantiske feil, logiske feil, kompileringsfeil og kjøretidsfeil. Syntaksfeil er kompileringsfeil og vanligvis enkle å rette opp. Kjøretidsfeil kan vi kontrollere med unntakshåndtering ([try-setninger](#)). Testing hjelper oss å finne logiske feil, og feilsøkerprogram (*Debugger*) hjelper oss å forstå og rette disse feilene. Vi har lært å skrive enkle tester i Python med [assert](#) og å bruke Visual Studio Code sin *debugger* til å stoppe utførelsen midt i et program for å inspisere koden og verdien til variabler når feilen inntreffer, [KM06](#).

generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

Å kapsle kode inn i funksjoner er første steg på veien til å generalisere løsninger. Arv er nok en metode for å lage gjenbrukbar programkode, [KM07](#).

vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

De første programmene vi har laget har, har ikke vært veldig brukervennlig. Vi har kun benyttet [input](#)- og [print](#)-funksjonene i Python for å kommunisere med brukeren. Vi har lært å formater tekst (Python [f-strings](#)) for bedre design ([KM03](#)) og brukervennlighet ([KM08](#)), og vi har erfart at programmer kan krasje. Det er ikke veldig brukervennlig så vi har brukt to metoder for å unngå disse problemene ([if](#)-setninger og [try](#)-setninger).

velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

Når vi skal samarbeide med andre, bør programkoden være enkel å forstå, se [KM09](#). Fornuftige variabel- og funksjonsnavn og gode kommentarer er med på å gjøre denne jobben lettere. Vi har også blitt kjent med [the Zen of Python](#) ([import this](#)), som er retningslinjer for hvordan vi bør designe programmene våre. Dessuten er UML diagrammer godt egnet til å utveksle idéer i designfasen.

gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

Vi har lest fra og skrevet til filer, blitt kjent med filsignaturer ([.jpg](#) og [.mp3](#)) og filtypene [.txt](#) og [.csv](#) for utveksling av data, [KM10](#).

bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

Det finnes mange reelle datasett gratis tilgjengelig på nettet, såkalte åpne data. Vi tok utgangspunkt i Felles Datakatalog og brukte kode til å innhente (laste ned) en [.csv](#)-fil. Dataene i csv-filen la vi inn i en tabell (liste med lister) samtidig som vi "vasket" og konverterte dem ([str](#) til [int](#)). Dermed ble det mulig å sortere dataene for å finne radene med de høyeste og laveste verdiene. Vi så at koden kunne forenkles med bruk av biblioteket [csv](#) og forenkles enda mer med bruk av [pandas](#). Til slutt presenterte vi dataene i et boksplot som ga et grafisk sammendrag av fordelingen til dataene, [KM11](#). Vi har også programmert innhenting, analysert og presentert informasjon grafisk fra Statistisk sentralbyrå.

## Runde 2

### 2.1 Grafisk brukergrensesnitt med tkinter

#### Bolkens innhold

Innføring i tkinter.....	62
Bilder, utseende og formatering.....	63
Spinbox.....	65
Radioknapper og avhukingsbokser.....	66
Listbox med rullefelt.....	67
Combobox.....	68
Objektorientert mal for tkinter.....	69
Ta med minst dette.....	70
Øvingsoppgaver.....	70

#### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

#### Innføring i tkinter

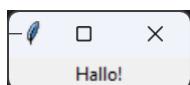
Det finnes flere [alternative pakker](#) for Python når vi skal lage vindusbaserte programmer med grafiske komponenter. Vi skal bruke [tkinter](#) som er enkelt å bruke, krever ikke så mye kode og passer på et nybegynner- og middels nivå. Dessuten kommer tkinter forhåndsmontert med Python, er plattformuavhengig (Windows, Mac, Unix), er gjennomprøvd og stabil programvare og det finnes mange tkinter-utvidelser.

Rammeverket for et GUI-program med tkinter ser ut som i koden til høyre. Tk-objektet `root` er hovedvinduet, og `root.mainloop()` starter hovedløkken som holder vinduet åpent og lytter etter hendelser som museklikk eller tastetrykk.

```
import tkinter as tk
root = tk.Tk()
root.mainloop()
```

Alt i tkinter er *widgets*; *Label*, *Entry*, *Button*, *Checkbutton*, osv. I programmet under oppretter vi en [Label](#)-widget i `root`-vinduet og legger den ut på skjermen med [.pack\(\)](#).

```
import tkinter as tk
root = tk.Tk()
tk.Label(root, text="Hallo!").pack()
root.mainloop()
```

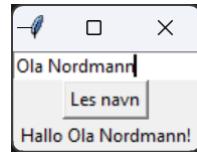
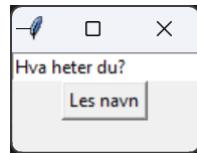


# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

La oss lage et program som leser inn et navn og bruker det i en utskrift.

```
1  import tkinter as tk
2
3
4  def skriv_ut():
5      hilsen.configure(text=f"Hello {navn.get()}!")
6
7
8  root = tk.Tk()
9
10 navn = tk.Entry(root)
11 navn.insert(0, "Hva heter du? ")
12 navn.pack()
13
14 tk.Button(root, text="Les navn", command=skriv_ut).pack()
15
16 hilsen = tk.Label(root)
17 hilsen.pack()
18
19 root.mainloop()
```



I linje 10 oppretter vi et inndatafelt ([Entry](#)-widget) i hovedvinduet (`root`). På grunn av at `.pack()` returnerer `None`, kan vi ikke legge *widgeten* ut på skjermen og samtidig beholde en referanse til komponenten i samme linje. I linje 11 setter vi inn en midlertidig forklarende tekst med `.insert()`, og til slutt viser vi komponenten på skjermen med `.pack()` i linje 12.

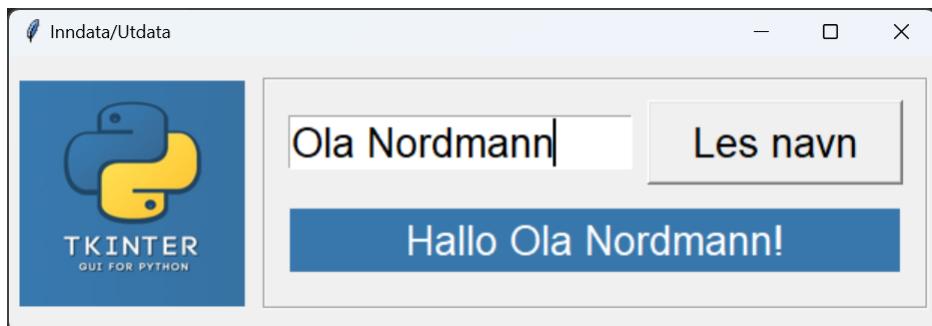
I linje 14 oppretter vi en knapp ([Button](#)-widget). Vi gir den en tekst med `text=`, kobler knappen til en funksjon med `command=` og viser knappen på skjermen med `.pack()`. Hvis funksjonen krever argumenter, bør vi bruke en lambda-funksjon, som for eksempel `command=lambda: min_funksjon('mitt_argument')`.

I linje 16 oppretter vi utdatafeltet ([Label](#)-widget) med en tom tekst og tilordner det til variabelen `hilsen`. I linje 17 viser vi utdatafeltet ut på skjermen med `.pack()`.

Når vi klikker på knappen kalles funksjonen `skriv_ut()` i linje 4. I linje 5 endrer vi på teksten i utdatafeltet med `.configure()`. Vi bruker f-streng og henter teksten fra inndatafeltet med `.get()`.

### Bilder, utseende og formatering

Det første programmet kan vel ikke sies å være brukervennlig med tanke på utseende, så la oss se på noen formateringsmuligheter. Utforsk koden nedenfor.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

```
import tkinter as tk           # font er en submodul,
from tkinter import font      # som ikke kan kalles direkte med tk.
from PIL import Image, ImageTk # husk pip install Pillow

# Funksjon for å vise hilsen basert på inndata
def skriv_ut():
    hilsen.configure(text=f"Hello {navn.get()}!")

root = tk.Tk()

# Endre standard skrift, gjelder ikke Entry-widgetet
default_font = tk.font.nametofont("TkDefaultFont")
default_font.configure(family="Arial", size=20)

# Endre tekst på tittellinjen
root.title("Inndata/Utdata")

# Legg til et bilde, krever from PIL import Image, ImageTk
bilde = Image.open("tkinter.jpg")
bilde = bilde.resize((150, 150), Image.Resampling.LANCZOS)
bilde = ImageTk.PhotoImage(bilde)
tk.Label(root, image=bilde).grid(row=0, column=0, padx=5, pady=5)

# Lag en ramme for høyre del
frame = tk.Frame(root, padx=10, pady=10)
frame.grid(row=0, column=1, padx=5, pady=15)

navn = tk.Entry(frame, width=15, font=("Arial 20"))
navn.insert(0, "Hva heter du? ")
navn.grid(row=0, column=0, padx=5, pady=13)

tk.Button(frame, text="Les navn", command=skriv_ut, width=10) \
    .grid(row=0, column=1, padx=5)

hilsen = tk.Label(frame, width=25, bg="#3877ac", fg="white", pady=3)
hilsen.grid(row=1, column=0, columnspan=2, pady=13)

root.mainloop()
```

Vi endrer på teksten på tittellinjen, og justerer standard skrift-type og -størrelse. Koden for å sette inn et bilde krever bruk av `Pillow`-biblioteket, som gir funksjonalitet for bildebehandling og grafikk. `Pillow` er en utvidelse av `PIL`-biblioteket og må først installeres med `pip install pillow`. Å inkludere et bilde i en `tkinter`-applikasjon tar 3-4 kodelinjer.

Tidligere så vi hvordan `.pack()` kan brukes for å vise komponenter på skjermen. Denne metoden er enkel og legger ut komponentene etter hverandre, enten vertikalt eller horisontalt, angitt med `side`-parameteren (*left, right, top, bottom*).

For mer komplekse layouter kan vi bruke `.grid()`, hvor vi plasserer *widgetene* i et rutenett. Hver celle justeres etter størrelsen på den største *widgeten* i raden eller kolonnen. Både `.pack()` og `.grid()` tilpasser automatisk størrelsen til *widgetenes* behov, noe som er bra, men ikke nødvendigvis ser fint ut. For et mer brukervennlig utseende, og for å gi luft rundt *widgetene*, kan vi bruke `padx` (horisontalt) og `pady` (vertikalt). Vi kan også oppgi bredde med `width`, velge for- og bakgrunnsfarge med `fg` og `bg`, med mer.

Det finnes også en metode, `.place()`, for nøyaktig posisjonering ved å spesifisere koordinater. Vi bør ikke kombinere disse metodene innenfor samme vindu eller ramme (`Frame`-widget).

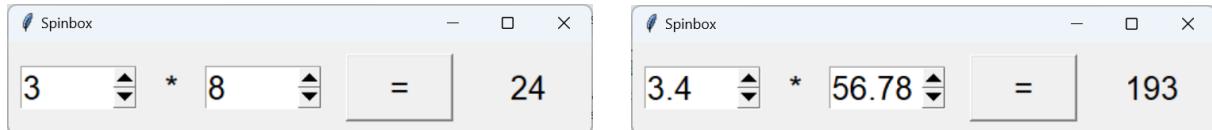
Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Spinbox

For å lese inn tall kan vi benytte oss av en **Spinbox** i stedet for **Entry**. Spinbox gir brukeren muligheten til å velge tall fra en forhåndsdefinert rekke, men tillater også manuell inntasting. Dette gir en mer brukervennlig opplevelse, men det kan også medføre ugyldig input. Derfor bruker vi unntaksbehandling for å sikre at inndataene er gyldige tall.



```
import tkinter as tk
from tkinter import font

def gange():
    try:
        svar = float(tall_1.get()) * float(tall_2.get())
    except ValueError:
        tall_1.delete(0, tk.END)
        tall_2.delete(0, tk.END)
        resultat.configure(text="")
    else:
        resultat.configure(text=f"{svar:.0f}")

root = tk.Tk()
root.title("Spinbox")

default_font = font.nametofont("TkDefaultFont")
default_font.configure(family="Arial", size=20)

tall_1 = tk.Spinbox(root, from_=0, to=10, increment=1,
                    width=5, font=("Arial 20"))
tall_1.grid(row=0, column=0, padx=10, pady=10)

gangetegn = tk.Label(root, text="*")
gangetegn.grid(row=0, column=1, padx=10, pady=10)

tall_2 = tk.Spinbox(root, from_=0, to=10, increment=1,
                    width=5, font=("Arial 20"))
tall_2.grid(row=0, column=2, padx=10, pady=10)

tk.Button(root, text="=", command=gange, width=5) \
    .grid(row=0, column=3, padx=10, pady=10)

resultat = tk.Label(root, width=5)
resultat.grid(row=0, column=4, padx=10, pady=10)

root.mainloop()
```

For å øke brukervennligheten ytterligere, ville det vært fordelaktig å inkludere et statusfelt som informerer brukeren om feil eller ugyldig input. Det ville også vært nyttig å håndtere vanlige inndatafeil, som å automatisk erstatte komma med punktum for desimaltall, så lenge vi ikke benytter `locale`-funksjoner (se [Språkinnstilling](#))

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Radioknapper og avhukingsbokser

I GUI-utvikling brukes radioknapper og avhukingsbokser for å la brukeren ta valg.

Radioknapper tillater kun én valgt verdi fra en gruppe, mens avhukingsbokser gir muligheten til å velge flere verdier.

Når vi ser nærmere på radioknapper i koden nedenfor, ser vi at vi først oppretter en variabel typen `IntVar()`, `DoubleVar()`, `StringVar()` eller `BooleanVar()` er også tillatte alternativer. Denne variablene lagrer verdien til den valgte radioknappen. Alle radioknappene i samme gruppe må knyttes til denne variablene, og hver radioknapp må ha sin egen distinkte verdi. `command=` brukes for å angi hvilken funksjon som skal utføres når en radioknapp er valgt. Parameteren `anchor=` justerer teksten i forhold til himmelretningene. `width=` må spesifiseres for at `anchor` skal fungere. Vi har også tatt med bruken av `enumerate` i `for`-løkker. `enumerate(farger)` gir oss både indeksen og verdien til elementene i listen farger. Dette gjør det enklere å lage radioknappene uten å måtte spesifisere hver verdi manuelt eller introdusere vår egen teller. `sticky`-parameteren bestemmer hvilken kant i cellen `widgeten` skal klistre seg til når den plasseres med `grid()`. Ved å kombinere retninger som `EW` eller `NS`, vil `widgeten` strekke seg for å fylle cellens bredde eller høyde.



```
import tkinter as tk
from tkinter import font

def velg_farge():
    idx_farge = farge_var.get()
    if idx_farge != -1:
        valg.configure(text=farger[idx_farge],
                      fg="white", bg=farger[idx_farge])

root = tk.Tk()
farger = ["red", "blue", "green"]
root.title("Radioknapper")
default_font = font.nametofont("TkDefaultFont")
default_font.configure(family="Arial", size=15)
farge_var = tk.IntVar(value=-1) # Ingen valgt

tk.Label(root, text="Velg farge:", bg="darkgrey", width=20,
         padx=5, pady=5).grid(row=0, column=0, padx=5, pady=5)

for idx, farge in enumerate(farger):
    tk.Radiobutton(root, text=farge, variable=farge_var, value=idx,
                  command=velg_farge, width=10, anchor="w") \
        .grid(row=idx + 1, column=0)

valg = tk.Label(root, text="")
valg.grid(row=4, column=0, padx=5, pady=5, sticky="NSEW")

root.mainloop()
```

Avhukingsbokser (`Checkbutton`) kodes nesten som radioknapper, men hver boks må ha sin egen unike variabel, noe som tillater flere samtidige valg, se eksempelvis <https://pythonbasics.org/tkinter-checkbox/> eller den vedlagte filen `b_2_1_6_radio_and_check.py`. Sistnevnte program benytter seg av `zip()`-funksjonen som kombinerer elementer fra to eller

# Smidig IT-2

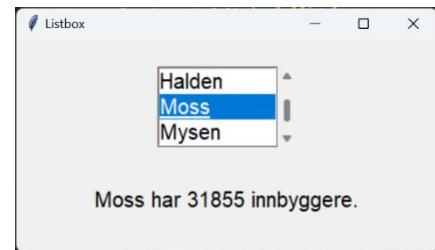
## for eksklusiv bruk ved <navn på skole>

flere, slik at vi får par (eller tupler) av elementer fra de opprinnelige listene. Dette vil vi lære mer om i neste bokl [2.2 Tupler og ordbøker](#).

### Listbox med rullefelt

I dette eksempelet utforsker vi `Listbox` i Tkinter hvor brukeren kan velge en by fra en liste for å få informasjon om byens befolkning. Vi legger til et rullefelt (`Scrollbar`) for å kunne navigere gjennom listen, ettersom det er flere byer enn det er plass til i området til listeboksen.

1. Først oppretter vi en `Frame` som skal inneholde både listeboksen og rullefeltet. Denne rammen blir plassert i hovedvinduet med litt vertikal *padding* for å sentrere den i vinduet.
2. I inni denne rammen oppretter vi et rullefelt (`Scrollbar`) med vertikal orientering.
3. Deretter oppretter vi `Listbox` inni samme ramme og kobler den til rullefeltet gjennom `yscrollcommand`. Med `selectmode="single"` sørger vi for at kun én by kan velges om gangen. Byene legges deretter til i listeboksen ved hjelp av `insert()`-metoden.
4. Rullefeltet må også konfigureres til å samspille med listeboksen ved bruk av `command=listbox.yview`.
5. Når brukeren utfører handlinger som å trykke på en tast eller klikke på en knapp, registreres dette som hendelser vi kan koble til funksjoner med `bind()`-metoden. Når hendelsen inntreffer, kjører funksjonen. Her kalles `vis_valgt_by()` hver gang et element i listen blir valgt (klikket på).
6. I funksjonen `vis_valgt_by()` henter vi først indeksen for valgt by med `curselection()`. Med denne indeksen henter vi både byens navn og innbyggertallet fra den todimensjonale listen `by_data`. Deretter lager vi en melding for å vise hvor mange innbyggere den valgte byen har. Denne meldingen vises på skjermen ved hjelp av en `Label` kalt `tekst_ut`.



```
import tkinter as tk
from tkinter import font

def vis_valgt_by(_):
    idx = listbox.curselection()[0]
    valgt_by = by_data[0][idx]
    innbyggere = by_data[1][idx]
    melding = f"{valgt_by} har {innbyggere} innbyggere."
    tekst_ut.configure(text=melding)

root = tk.Tk()
root.geometry("400x200")
root.title("Listbox")
default_font = font.nametofont("TkDefaultFont")
default_font.configure(family="Arial", size=15)
by_data = [[["Askim", "Fredrikstad", "Halden",
            "Moss", "Mysen", "Sarpsborg"], [14259, 65415, 25300, 31855, 6513, 45852]]]

ramme = tk.Frame(root)
scrollbar = tk.Scrollbar(ramme, orient="vertical")
listbox = tk.Listbox(ramme, height=3, width=10,
                     yscrollcommand=scrollbar.set,
                     selectmode="single")
for by in by_data[0]:
    listbox.insert("end", by)

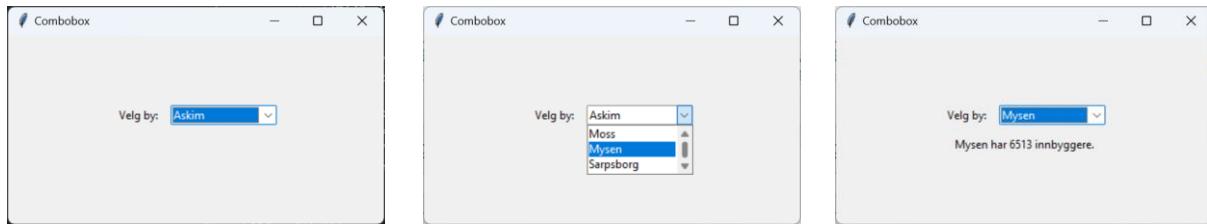
ramme.pack(pady=25)
scrollbar.configure(command=listbox.yview)
scrollbar.pack(side="right", fill="y")
listbox.pack()
listbox.bind("<<ListboxSelect>>", vis_valgt_by)
tekst_ut = tk.Label(root, text="")
tekst_ut.pack(pady=10)

root.mainloop()
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Combobox



En **Combobox** fungerer som en rullegardinliste som opptar mindre plass enn en **Listbox**, som kan vise flere alternativer hele tiden. **Tk**-pakken inneholder ingen **Combobox**, så vi bruker **ttk**-modulen. **ttk-widgets** ser noe mer moderne ut, bruker stiler og er noe vanskeligere å formatere enn standard **Tk-widgets**. **Combobox** håndterer imidlertid rullefelt automatisk. I eksempelet, har vi også brukt **trace\_add**, som krever Python 3.8 eller nyere, og som reagerer på endringer i kontrollvariabelen. Dette er en enklere, renere og mer fleksibel metode enn **bind**, fordi flere uavhengige funksjoner kan reagere på variabelendringene.

```
1  import tkinter as tk
2  from tkinter import ttk
3
4  by_data = [[["Askim", "Fredrikstad", "Halden", "Moss", "Mysen", "Sarpsborg"],
5  |           |           |           |           |           |
6  |           [14259, 65415, 25300, 31855, 6513, 45852]]
7
8  def oppdater_tekst(*args):
9      idx = combobox.current()
10     if idx >= 0: # Sjekker at en gyldig indeks er valgt
11         valgt_by = by_data[0][idx]
12         innbyggere = by_data[1][idx]
13         melding = f"{valgt_by} har {innbyggere} innbyggere."
14         tekst_ut.configure(text=melding)
15
16
17 root = tk.Tk()
18 root.geometry("400x200")
19 root.title("Combobox")
20 # Plasser alt i en sentrert ramme
21 frame = ttk.Frame(root)
22 frame.place(relx=0.5, rely=0.5, anchor='center')
23 # Label for byvalt
24 tk.Label(frame, text="Velg by:").grid(row=0, column=0, padx=5, pady=5)
25 # Opprett Comboboxen
26 valgt_by = tk.StringVar()
27 combobox = ttk.Combobox(frame, textvariable=valgt_by,
28 |           |           |           |           |
29 |           |           |           |           |           values=by_data[0], state="readonly", width=15)
30 combobox.grid(row=0, column=1, padx=5, pady=5)
31 combobox.current(0) # Setter standardvalg
32 combobox.configure(height=3) # for å demonstere rullefelt
33 valgt_by.trace_add("write", oppdater_tekst) # Følg med på endringer
34 # Utdata
35 tekst_ut = tk.Label(frame, text="")
36 tekst_ut.grid(row=1, column=0, columnspan=2, padx=5, pady=5)
37 root.mainloop()
```

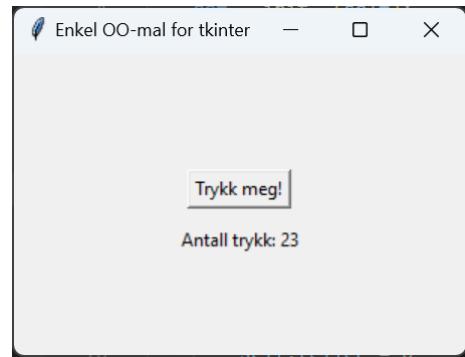
Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Objektorientert mal for tkinter

På samme måte som mange av programmene våre er bygget opp med klasser og objekter, gjelder dette også for brukergrensesnittet med `tkinter`. Alle komponentene, som `Label`, `Button`, og `Entry`, er objekter. Mer presist, de er underklasser som arver metoder og egenskaper fra en overordnet klasse kalt `Widget`. Hovedvinduet, ofte referert til som `root`, er en instans av klassen `Tk` og kan inneholde objekter av typen `Widget`. Så hver gang vi jobber med disse komponentene i `tkinter`, benytter vi prinsippene fra objektorientert programmering.



Det er ikke mye som skal til for å gjøre hele programmet objektorientert. Her har vi laget en enkel mal for objektorienterte tkinter-programmer. Vi samlar logikken og brukergrensesnittet i en klasse kalt `App`. Senere skal vi lære hvordan vi kan lage separate klasser for logikken og brukergrensesnittet, noe som gir enda flere fordeler.

Det er ikke nødvendig å forstå alle detaljene i OO-malen for tkinter nå, for mye vil falle på plass etter [2.6 Objektorientert Programmering](#). `__init__` er en metode som kalles automatisk når et nytt objekt av klassen opprettes. `self` er en referanse til objektet "innenfra". Med `self` får vi tak i det spesifikke objektets egenskaper og metoder. `if __name__ == "__main__":` brukes for å sikre at koden bare kjøres når vi starter denne spesifikke filen, og ikke hvis vi importerer denne filen i et annet program.

```
import tkinter as tk

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Enkel OO-mal for tkinter")
        self.root.geometry("300x200")
        self.frame = tk.Frame(self.root)
        self.frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self.button = tk.Button(
            self.frame, text="Trykk meg!", command=self.on_click
        )
        self.button.pack(pady=10)
        self.label = tk.Label(self.frame, text="Antall trykk: 0")
        self.label.pack()
        self.teller = 0

    def on_click(self):
        self.teller += 1
        self.label.config(text=f"Antall trykk: {self.teller}")

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = App()
    app.run()
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

**Tkinter** er et brukervennlig og plattformuavhengig bibliotek for å lage grafiske brukergrensesnitt. `root = tkinter.Tk()` oppretter hovedvinduet, og `root.mainloop()` starter hovedløkken som holder vinduet åpent og lytter etter hendelser som museklikk eller tastetrykk. Grafiske komponenter som `Button`, `Label` (utdata), `Entry` (inndata), `Spinbox`, `Radiobutton`, `Checkbutton` og `Listbox` kalles *widgets*. Disse plasseres i vinduet med layout-metoder som `pack()`, `grid()` og `place()`. Noen *widgets*, som `Button`, `Radiobutton` og `Checkbutton`, har en `command`-parameter hvor vi kan angi en funksjon som kalles når noen klikker på objektet. Det er også mulig å fange opp hendelser ved å knytte dem til objekter med metoden `bind`, se [Vedlegg 5.6 Hendelser og animasjoner i tkinter](#). Når vi senere skal programmere animasjoner og håndtere kollisjoner, har vi valgt å bruke **Pygame**, som er et annet GUI-bibliotek. Da vil vi lære mer om hendelsesstyring, som utføres på en litt annen måte i Pygame enn i tkinter.

### Øvingsoppgaver

Variasjon: I oppgave 2.1.1 kan du bruke `pack()` for å legge ut komponentene, så kan du bruke `grid()` i oppgave 2.1.2

#### 2.1.1

Ta for deg oppgave 1.7.3 (og 1.3.2) som oversetter vanlige tall (titallsystemet) til binære tall. Løs oppgaven med GUI. Ta gjerne utgangspunkt i løsningsforslaget.

```
while True:
    tekst = input('Skriv inn et heltall: ')

    if tekst == 'slutt':
        print('\nTakk for nå.')
        break

    try:
        tall = int(tekst)
    except ValueError:
        print(f'{tekst} er ikke et heltall.')
    else:
        print(f'{tall} i titallsystemet er {tall:b} binært.')
    finally:
        print('\nVi fortsetter. Skriv "slutt" for å avslutte.')
```

#### 2.1.2

Ta for deg oppgave 1.2.1<sup>13</sup> om beregning av flytid og løs oppgaven med GUI. Ta gjerne utgangspunkt i løsningsforslaget og sett inn bildet av hastighetsmåler hvis du vil.

```
import datetime

avstand_i_km = float(input('Oppgi avstand i km: '))
fart_i_knop = float(input('Oppgi fart i knop: '))
fart_i_kmpt = fart_i_knop * 1.852
flytid = avstand_i_km / fart_i_kmpt

tid = datetime.timedelta(hours=flytid) # 'hh:mm:ss'
tid = str(tid).split(':')           # ['hh', 'mm', 'ss']
print(f'Timer: {tid[0]}, minutter: {tid[1]}, sekunder: {tid[2][:2]}')
```

<sup>13</sup> Fra IT-2 eksamen våren 2010 (LK06)

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

2.1.3

Løs oppgave 2b, IT-2 Eksamens høsten 2019<sup>14</sup> (LK06).

På [Python Tutorial](#)-nettsidene finnes informasjon om Tkinter [Combobox](#).

### Oppgave 2, IT-2 Eksamens høsten 2019 (LK06)

Du skal lage en applikasjon som beregner kaloriforbruk for en gitt aktivitet – en slags treningskalkulator. Beregningen skal basere seg på valgt **aktivitet** i kombinasjon med valgt **intensitet** og **varighet** på treningen.

Krav:

- Brukeren skal kunne velge mellom disse fem aktivitetene:
  - Aerobics (814 kcal/time)
  - Bordtennis (236 kcal/time)
  - Fotball (510 kcal/time)
  - Golf (244 kcal/time)
  - Jogging (666 kcal/time)
- Brukeren skal kunne velge mellom disse intensitetsnivåene:
  - Lavt (Du kan gange kaloriforbruket med 0,8 for å trekke fra 20 %.)
  - Middels (Kaloriforbruket som er oppgitt.)
  - Høyt (Du kan gange kaloriforbruket med 1,2 for å legge til 20 %.)
  - Brukeren skal kunne oppgi varighet i minutter.

(Tips: Én time er 60 minutter. Du må dele oppgitt varighet i minutter på 60 for å få antall timer.)

- Løsningen skal implementeres etter denne grensesnittskissen (wireframe):
  - Brukeren skal velge aktivitet fra en nedtrekksliste.
  - Brukeren skal velge intensitet på treningen ved å velge én av tre radiobuttons.
  - Brukeren skal oppgi varighet på treningen i minutter i et tekstfelt.
  - Ved klick på knappen skal kaloriforbruket for valgt aktivitet, intensitet og angitt varighet beregnes og vises.

Figur 4: Brukergrensesnitt for applikasjonen

#### Oppgave

- a. Lag et flytdiagram for applikasjonen etter kravene.
- b. Lag applikasjonen etter kravene.

<sup>14</sup> Kun for de som ønsker flere utfordringer

## 2.2 Tupler og ordbøker

### Bolkens innhold

Tupler .....	72
Ordbøker.....	74
Ta med minst dette.....	78
Øvingsoppgaver.....	78

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

### Tupler

I bolken [1.5 Lister](#) ble vi kjent med datatypen `list`, som kan lagre flere verdier. Disse verdiene, eller objektene, trenger ikke å være av samme type. Vi kan få tilgang til et spesifikt objekt i listen ved å bruke indekstallet (posisjonen) til elementet. Videre finnes det flere metoder for å endre listen. En tuppel er også en sammensatt datatype som kan inneholde flere verdier. Den store forskjellen er at mens vi kan endre en liste etter den er opprettet, kan vi ikke endre en tuppel (*immutable*). Den angis med vanlige parenteser `( )`, i motsetning til hakparenteser `[ ]` som brukes for lister. Hvis vi ikke trenger å endre innholdet i en liste, kan vi bruke en `tuple` i stedet. Tuppelen tar mindre plass, utføres raskere og bidrar til å unngå utilsiktede feil. Her er et eksempel som demonstrerer noen av egenskapene til tupler.

```
farger = ('rød', 'grønn', 'blå')
print(type(farger))
print(farger)
for farge in farger:
    print(f'{farge.upper()} ', end='')
print('\n' + farger[1].capitalize())
farger[1]='gul' # TypeError: 'tuple' object does not support item assignment
✖ 0.3s
<class 'tuple'>
('rød', 'grønn', 'blå')
RØD GRØNN BLÅ
Grønn
```

Hvis vi dropper parentesene, blir det også en tuppel. Videre kan vi tilordne flere variabler samtidig ved å pakke ut verdier fra en tuppel (*multiple assignment, tuple unpacking, eller iterable unpacking*).

```
farger = 'rød', 'grønn', 'blå'
x, y , z = farger
print(x, y, z)

✓ 0.0s
rød grønn blå
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

For eksempel, i oppgavene der vi beregnet flytid, kunne vi ha forenklet og forbedret lesbarheten ved å bruke variabeltilordning. I stedet for å henvise til individuelle indekser, kan vi tilordne verdier direkte til variablene.

```
import datetime
flytid = 1.23
tid = datetime.timedelta(hours=flytid) # 'hh:mm:ss'
hh, mm, ss = str(tid).split(':')      # ['hh', 'mm', 'ss']
print(f'Timer: {hh}, minutter: {mm}, sekunder: {ss[:2]}')
✓ 0.0s
Timer: 1, minutter: 13, sekunder: 48
```

Metoden med å pakke ut verdier fra en tuppel eller liste kan brukes til å bytte verdier mellom to variabler uten en midlertidig variabel, som ofte kreves i andre programmeringsspråk.

Denne teknikken kunne vi for eksempel brukt i oppgavene om Fibonaccitallene ( $f_1, f_2 = f_2, f_1 + f_2$ ).

```
a=1; b=2;           print(f'a={a} og b={b}')
tmp = a; a=b; b=tmp; print(f'a={a} og b={b}')
c=3; d=4;           print(f'c={c} og d={d}')
c,d = d,c;         print(f'c={c} og d={d}')
✓ 0.0s
a=1 og b=2
a=2 og b=1
c=3 og d=4
c=4 og d=3
```

En slik tilordning av variabler gjelder for alle datatyper som kan gjennomløpes med en **for-løkke (iterables)**, eksempelvis `a, b, c = "XYZ"`, hvor `a`, `b` og `c` blir henholdsvis `X`, `Y` og `Z`, eller `d, e, f = range(1, 4)`, hvor `d`, `e` og `f` blir henholdsvis `1`, `2` og `3`.

En annen hendig funksjon i Python er stjerneoperatøren `*` som kan brukes til å pakke ut elementene i en tuppel eller liste. Dette betyr at i stedet for å skrive ut hele tuppelen eller listen som en enkelt enhet, eller bruke en **for-løkke**, vil elementene skrives ut som individuelle verdier. Når vi bruker `*` i forbindelse med `print()`, skrives hvert element i tuppelen eller listen som separate argumenter.

```
farger = 'rød', 'grønn', 'blå'
print(*farger)
✓ 0.0s
rød grønn blå
```

`-operatoren` kan også brukes i variabeltilordning for å håndtere flere verdier fra en tuppel eller liste. Her tilordnes først `a` og `b` til de to første verdiene og deretter `d` til den siste verdien. Til slutt tilordnes de resterende verdiene som en liste til `c`, som er angitt med `*`. Det sier seg selv at det bare kan være én `-operator` i en slik variabeltilordning.

```
farger = ('gul', 'oransje', 'rød', 'lilla', 'blå', 'grønn')
a, b, *c, d = farger
print(a, b, c, d, sep='\n')
✓ 0.0s
gul
oransje
['rød', 'lilla', 'blå']
grønn
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi ønsker å hente ut verdier fra to eller flere lister, kan vi bruke `zip()`. Den returnerer et objekt av typen `zip` som inneholder tupler med parvise verdier fra de originale listene og kan gjennomløpes i en for-løkke.

```
produkter = ["Eple", "Banan", "Appelsin"]
priser = [10, 8, 13]
kombinert = zip(produkter, priser)
print(kombinert)
print(*kombinert)

for produkt, pris in zip(produkter, priser):
    print(f'{produkt} koster {pris} kr')
✓ 0.0s

<zip object at 0x00000181D5C07B00>
('Eple', 10) ('Banan', 8) ('Appelsin', 13)
Eple koster 10 kr
Banan koster 8 kr
Appelsin koster 13 kr
```

### Ordbøker

En ordbok (*dictionary*) kalles også assosiative tabeller, der vi bruker navn i stedet for indekserte tall på elementene. Hvert element består av et nøkkel/verdi-par (*key-value pair*). Nøkklene må være unike. Datatypen for ordbøker heter `dict`, og vi oppretter dem ved å bruke krøllparenteser `{ }`. For å få tilgang til elementene i ordboken, bruker vi nøkkelen innenfor firkantede parenteser. Det kan brukes til å lese, endre, legge til og slette verdier i ordboken.

```
ordbok = {'fornavn': 'Per', 'poststed': 'Askim'}
print(type(ordbok))
print(ordbok)
print(ordbok['poststed']) # Skriv ut poststed
ordbok['fornavn'] = 'Pål' # Endre fornavn
print(ordbok)
ordbok['fødselsår'] = 1992 # Legg til fødselsår
print(ordbok)
del ordbok['poststed'] # Slett poststed
print(ordbok)
✓ 0.0s

<class 'dict'>
{'fornavn': 'Per', 'poststed': 'Askim'}
Askim
{'fornavn': 'Pål', 'poststed': 'Askim'}
{'fornavn': 'Pål', 'poststed': 'Askim', 'fødselsår': 1992}
{'fornavn': 'Pål', 'fødselsår': 1992}
```

Vi kan gjennomløpe en ordbok med en `for`-løkke ved hjelp av metoder som `keys()`, `values()`, og `items()`. Bruk av `keys()` gir oss bare nøkklene fra ordboken (standard), `values()` gir oss bare verdiene,

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
ordbok = {  
    'fornavn': 'Per',  
    'yrke': 'Snekker',  
    'fødselsår': 1992,  
    'poststed': 'Askim'  
}  
  
print('Nøkler:')  
for nøkkel in ordbok.keys():  
    print(nøkkel,end=' ')  
✓ 0.0s  
  
Nøkler:  
fornavn yrke fødselsår poststed  
  
print('Verdier:')  
for verdi in ordbok.values():  
    print(verdi,end=' ')  
✓ 0.0s  
  
Verdier:  
Per Snekker 1992 Askim
```

mens `items()` gir oss både nøklene og verdiene som par.

```
print('Nøkler og verdier:')  
for nøkkel,verdi in ordbok.items():  
    print(f'{nøkkel.capitalize()}: {verdi}')  
✓ 0.0s  
  
Nøkler og verdier:  
Fornavn: Per  
Yrke: Snekker  
Fødselsår: 1992  
Poststed: Askim
```

Vi kan lage en liste med ordbøker. Hvert element i listen inneholder informasjon om en person lagret som en ordbok. Tenk på radene som personer og kolonnene som nøklene. Vi kan finne den eldste og yngste personen ved å bruke funksjonene `max` og `min` sammen med `itemgetter` for å velge kolonnen `alder`.

```
personer = [  
    {'navn': 'Knut', 'alder': 32},  
    {'navn': 'Tiril', 'alder': 28},  
    {'navn': 'Anne', 'alder': 41},  
    {'navn': 'Oscar', 'alder': 35}  
]  
  
from operator import itemgetter  
eldst = max(personer, key=itemgetter('alder'))  
print(f'{eldst["navn"]} på {eldst["alder"]} er eldst.')  
yngst = min(personer, key=itemgetter('alder'))  
print(f'{yngst["navn"]} på {yngst["alder"]} er yngst.')  
✓ 0.0s  
  
Anne på 41 er eldst.  
Tiril på 28 er yngst.
```

Vurderingsseksempler

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Ved hjelp av `itemgetter()` kan vi sortere listen basert på en spesifikk nøkkel (eller kolonne), som for eksempel `navn`.

```
print('Sortert alfabetisk')
for person in sorted(personer, key = itemgetter('navn')):
    print(f'{person["navn"][:6]}{person["alder"][:3]} år')
print('\nOpprinnelig liste')
print('\n'.join(str(person) for person in personer))

✓ 0.0s

Sortert alfabetisk
Anne 41 år
Knut 32 år
Oscar 35 år
Tiril 28 år

Opprinnelig liste
{'navn': 'Knut', 'alder': 32}
{'navn': 'Tiril', 'alder': 28}
{'navn': 'Anne', 'alder': 41}
{'navn': 'Oscar', 'alder': 35}
```

Vi ble introdusert til pandasbiblioteket i bok 1.9 da vi jobbet med reelle datasett og CSV-filer. Senere skal vi se nærmere på pandas i bokene 3.1, 3.8 og 4.3. Nå tar vi kun med hvordan pandas kan sortere en liste med ordbøker. Med pandas kan vi enkelt omdanne listen til en `DataFrame`, som er en tabellstruktur, og deretter sortere den

```
print('Pandas:')
import pandas as pd
df = pd.DataFrame(personer).sort_values(by='navn')
print(df)

✓ 0.0s

Pandas:
   navn  alder
2  Anne    41
0  Knut    32
3  Oscar    35
1  Tiril   28
```

Det finnes ingen `Table-widget` i `tkinter`, men vi kan bruke løkker og legge ut `Label-widgeter` med `grid`, som vist i figuren til høyre og i filen `b_2_2_3_tabell_med_løkke.py`.

Navn	Alder
Knut	32
Tiril	28
Anne	41
Oscar	35

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

	navn	alder
1	Anne	41
2	Oscar	35
3	Knut	32
4	Tiril	28

Det er også mulig å bruke `ttk.Treeview`-widgeten som vist i [Python Tkinter TreeView – How to Use](#), men vi foretrekker en enklere løsning ved å bruke `pandastable`-biblioteket. Det må først installeres med `pip install pandastable`. Vi lager en `DataFrame` fra listen med ordbøker, hvor nøkkelnavnene blir til kolonneoverskrifter. Deretter kan vi lage en `Table` basert på denne `DataFrame`-en og presentere den som en tabell. Utforsk de mange mulighetene som `pandastable` byr på. For eksempel kan vi sortere data ved å klikke på kolonneoverskriftene, endre innhold i cellene direkte, filtrere rader og lage enkle diagrammer. Høyreklikk på tabellen for å se en meny med de forskjellige funksjonene som er tilgjengelige.

Ordbøker er lettere å vedlikeholde enn lister. Vi har tidligere sett at det kan føre til uforutsigbare resultater når vi sletter elementer fra en liste inni en for-løkke, fordi indeksene til de gjenværende elementene forskyves og lengden på listen endres. Dette problemet eksisterer ikke med ordbøker. Ordbøker gir dessuten raskere tilgang enn lister. Koden blir også mer lesbar ved å benytte tekstbaserte nøkler, i motsetning til tallbaserte indeks i lister. Ordbøker var tidligere uordnede, men fra Python 3.7 bevares rekkefølgen.

```
import tkinter as tk
import pandas as pd
from pandastable import Table

class App:
    def __init__(self):

        # Lag en DataFrame av listen
        df = pd.DataFrame([
            {'navn': 'Knut', 'alder': 32},
            {'navn': 'Tiril', 'alder': 28},
            {'navn': 'Anne', 'alder': 41},
            {'navn': 'Oscar', 'alder': 35}
        ])

        # Sett opp og midtstill vinduet
        self.root = tk.Tk()
        self.root.title("PandasTable")
        mid_x = self.root.winfo_screenwidth() // 2
        mid_y = self.root.winfo_screenheight() // 2
        self.root.geometry(f'250x150+{mid_x-125}+{mid_y-75}')

        # Lag en ramme til Table
        frame = tk.Frame(self.root)
        frame.pack(expand=True, fill=tk.BOTH)

        # Lag en Table av df i frame
        pt = Table(frame, dataframe=df)
        pt.show()

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = App()
    app.run()
```

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Det finnes også en fjerde grunnleggende datatype som kan inneholde flere verdier. Sett ([set](#)), som vi skal lære om i bok [3.3 Sett og boolsk algebra](#), kan ikke inneholde like verdier.

Datatype	list	tuple	dict	set
Syntaks	[ ]	( )	{ }	{ }
Rekkefølge	Ordnet	Ordnet	Ordnet (>=3.7)	<b>Uordnet</b>
Kan endres?	Ja	<b>Nei</b>	Ja	Ja
Duplikate verdier?	Ja	Ja	Ja	<b>Nei</b>
Oppslag	Indeks (heltall)	Indeks (heltall)	Nøkkel (tekst)	<b>Ingen</b> <sup>15</sup>
Kan gjennomløpes?	Ja	Ja	Ja	Ja

### Ta med minst dette

En tuppel er en datatype som kan inneholde en samling av verdier, som en liste, men den kan ikke endres etter at den er opprettet. I motsetning til lister som opprettes med hakeparenteser, opprettes tupler med vanlige parenteser. Hvis vi ikke bruker parenteser, men bare verdier adskilt med komma, blir det en tuppel. Tupler krever kortere prosesseringstid, brukes ofte for variabeltilordning og utpakking av verdier, som kan forenkle kode og forbedre lesbarheten. De kan også brukes til å bytte variabler, som i `a, b = b, a`.

Stjerneoperatøren (\*) kan brukes for å pakke ut elementene både i en tupler og lister.

Ordbøker er også en datatype som kan inneholde en samling av verdier, som en liste. Men i motsetning til lister som er basert på indeks, lagrer ordbøker data i nøkkel/verdi-par, hvor hver nøkkel er unik. Koden blir mer lesbar og gir raskere tilgang til data. Vi kan også lage løkker med ordbøker hvor vi henter ut alle nøklene ([keys](#)), verdiene ([values](#)) eller nøkkel/verdi-par ([items](#)). Ordbøker opprettes med krøllparenteser. Vi kan enkelt legge til et nytt nøkkel/verdi-par ved å tildele en verdi til en ny nøkkel, endre en eksisterende verdi ved å tildele en ny verdi til en eksisterende nøkkel, eller slette et par med [del](#)-kommandoen. Da bruker vi hakeparenteser med nøkkelverdien i anførselstegn som `ordbok["nøkkel"]`. Fra og med Python 3.7 blir rekkefølgen på elementene bevart. Tabeller, i form av lister med ordbøker, kan vi skrive ut i terminalvinduet, med [grid](#) i [tkinter](#) eller kanskje aller enklest ved å bruke [pandastable](#).

### Øvingsoppgaver

Bruk også det du har lært om lyster til å løse disse oppgavene med tupler.

#### 2.2.1

- Lag en tuppel med 'mor', 'far' og 'datter'.
- Skriv ut det første elementet i tuppelen.
- Skriv ut det siste elementet i tuppelen.

---

<sup>15</sup> Sett tillater ikke direkte oppslag basert på en indeks eller nøkkel, men vi kan bruke [in](#)-operatøren til å sjekke om en verdi er med i settet.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

2.2.2

- a) Lag en tuppel med tallene fra 1 til 5.
- b) Finn summen til tallene.
- c) Finn det høyeste og laveste tallet i tuppelen.

2.2.3

Gitt tuplene  $t1 = (1, 2, 3)$  og  $t2 = (4, 5, 6)$ , lag en ny tuppel  $t3$  som føyer sammen  $t1$  og  $t2$ .

2.2.4

- a) Lag en tuppel med navnene 'Linda', 'Mina', 'Nina', og 'Sanna'.
- b) Sjekk om 'Mina' finnes i tuppelen
- c) Sjekk indeksen til 'Nina' i tuppelen.

2.2.5

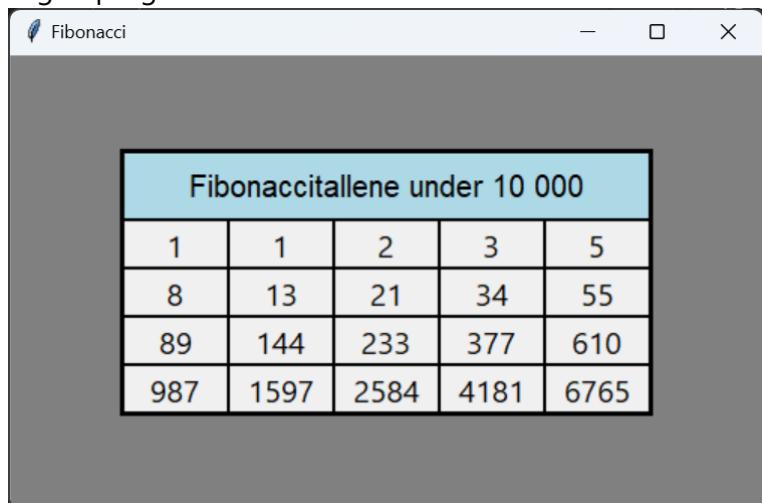
- a) Lag en tuppel med tallene fra 13 til og med 16.
- b) Konverter denne tuppelen til en liste.
- c) Legg til tallet 17 i listen.
- d) Konverter listen tilbake til en tuppel.

2.2.6

- a) Lag en tuppel med 5 elementer: 'Løve', 'Leopard', 'Elefant', 'Bøffel' og 'Neshorn'.
- b) Bruk utpakking for å tildele hvert av dyrene til en variabel.
- c) Skriv ut hver variabel.

2.2.7

Lag et program som viser Fibonaccitallene under 10 000 i en tabell som vist i figuren.



The screenshot shows a Windows application window titled "Fibonacci". The main content is a table with a light blue header row containing the text "Fibonaccitallene under 10 000". The table has five columns and four data rows. The data rows contain the following values:

1	1	2	3	5
8	13	21	34	55
89	144	233	377	610
987	1597	2584	4181	6765

2.2.8

- a) Lag en ordbok  $bok$  hvor nøkkelen  $tittel$  har verdien  $Markens grøde$ , nøkkelen  $forfatter$  har verdien  $Knut Hamsun$ , og nøkkelen  $år$  har verdien  $1917$ .
- b) Skriv ut tittelen på boka

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

c) Skriv ut forfatteren til boka

2.2.9

- Lag en ordbok **by** hvor nøkkelen **navn** har verdien **Trondheim**, nøkkelen **befolkning** har verdien **210496**, og nøkkelen **land** har verdien **Norge**.
- Legg til nøkkelen **språk** med verdien **norsk** til ordboken.
- Fjern nøkkelen **befolkning** fra ordboken.
- Skriv ut den oppdaterte ordboken.

2.2.10

Ta utgangspunkt i denne tabellen:

```
personer = [  
    {'fornavn': 'Kari', 'etternavn': 'Hansen', 'fødselsår': 2001},  
    {'fornavn': 'Gustav', 'etternavn': 'Monsen', 'fødselsår': 1995},  
    {'fornavn': 'Anette', 'etternavn': 'Ås', 'fødselsår': 1998},  
    {'fornavn': 'Marius', 'etternavn': 'Lie', 'fødselsår': 2002},  
    {'fornavn': 'Wenche', 'etternavn': 'Hovland', 'fødselsår': 1999}  
]
```

Se filen [o\\_2\\_2\\_10.py](#)

Lag et program hvor brukeren kan velge fra kommandolinjen hvordan tabellen skal sorteres.  
Se forslag nedenfor.

Fornavn	Etternavn	Fødselsår
Anette	Ås	1998
Gustav	Monsen	1995
Kari	Hansen	2001
Marius	Lie	2002
Wenche	Hovland	1999

Sorter etter følgende rekkefølge:  
F=fornavn, E=etternavn, A=Fødselsår,S=Slutt  
Velg F, E, A eller S:

Fornavn	Etternavn	Fødselsår
Kari	Hansen	2001
Wenche	Hovland	1999
Marius	Lie	2002
Gustav	Monsen	1995
Anette	Ås	1998

Sorter etter følgende rekkefølge:  
F=fornavn, E=etternavn, A=Fødselsår,S=Slutt  
Velg F, E, A eller S:

Fornavn	Etternavn	Fødselsår
Gustav	Monsen	1995
Anette	Ås	1998
Wenche	HovLand	1999
Kari	Hansen	2001
Marius	Lie	2002

Sorter etter følgende rekkefølge:  
F=fornavn, E=etternavn, A=Fødselsår,S=Slutt  
Velg F, E, A eller S:

[Takk for nå](#)

Koden under kan brukes til å blanke terminalvinduet etter hver innlesning:

```
import os  
  
if os.name == 'posix':    # macOS og Linux  
|   os.system('clear')  
else:  
|   os.system('cls')      # Windows
```

2.2.11

Lag et program med tilsvarende funksjonalitet som i 2.2.10 og med grafisk brukergrensesnitt.

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 2.3 Versjonskontroll med Git

### Bokens innhold

Git og GitHub.....	81
Git for personlig bruk fra kommandolinjen.....	82
Git-ordliste .....	83
Git for personlig bruk med VS Code.....	83
Ta med minst dette.....	84
Øvingsoppgaver.....	85

### Kompetansemål:

- vurdere og bruke strategier for feilsøking og testing av programkode
- velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

### Git og GitHub

[Git](#) er et versjonskontrollsyste som hjelper oss med å organisere og spore endringer i kode under programutvikling. Det gir oss flere fordeler, blant annet:

- sikkerhet mot feil med mulighet for gjenoppretting av tidligere versjoner
- godt egnet til samarbeid, hvor utviklere kan jobbe parallelt ([KM09](#))
- tillater sidegrener for eksperimentering og feilretting uten å påvirke hovedgrenen

I tillegg er Git gratis og åpen kildekode. Git kan brukes uavhengig av [GitHub](#), som er en skytjeneste som benytter Git og tilbyr flere nyttige funksjoner for utviklingsprosjekter. Mens GitHub gir gratis funksjonalitet for offentlige prosjekter, byr betalte versjoner på utvidet funksjonalitet. I januar 2023 hadde GitHub over 100 millioner registrerte utviklere og mer enn 372 millioner kodeoppbevaringer (*repositories*), hvorav minst 28 millioner av dem var offentlige ([Wikipedia](#)).

Før vi begynner å samarbeide med Git, skal vi først lære å anvende det hver for oss uten å ta i bruk GitHub. Git er et distribuert versjonskontrollsyste, som betyr at flere kan laste ned og arbeide med filene lokalt. Etter endringer kan filene lastes opp igjen og integreres i et sentralt oppbevaringssted. I denne runden skal vi begrense oss til å anvende Git lokalt for personlig bruk.

For å jobbe med Git kan vi enten bruke kommandolinjen eller en [Git-GUI-klient](#). Vi starter med å bruke PowerShell i Windows eller Terminal i macOS for å bli kjent med de grunnleggende kommandoene og arbeidsrutinene. Deretter går vi over til VS Code, som også fungerer som en GUI-klient. Selv om GUI-klienter ofte gjør Git enklere for oss siden vi slipper å huske alle kommandoene, må vi likevel forstå hvordan Git fungerer og være klar over begrensningene i GUI-verktøyet.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Git for personlig bruk fra kommandolinjen

Først, last ned Git fra <https://git-scm.com/downloads>. Mac-brukere kan skrive `git` i terminalvinduet for å bli bedt om å installere Xcode Command Line Tools, som inneholder Git. Under installasjonen får vi valget om hva vi vil kalle hovedgrenen. Tradisjonelt har denne blitt kalt *master*, men siden det kan forbindes med *slave*, har mange nå gått over til å bruke *main*, som også vi skal gjøre.

Åpne Terminal (eller PowerShell i Windows) og bekrefte at Git er installert ved å skrive:

- `git --version`

Angi brukernavn og e-postadresse:

- `git config --global user.name <ditt navn>`
- `git config --global user.email <din epost adr.>`

Sjekk at hovedgrenen er satt til *main*:

- `git config --global --get init.defaultBranch`

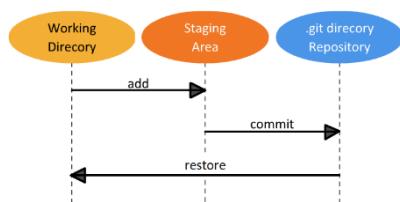
Om nødvendig, bytt til *main*:

- `git config --global init.defaultBranch main`

Kontroller inntastingen:

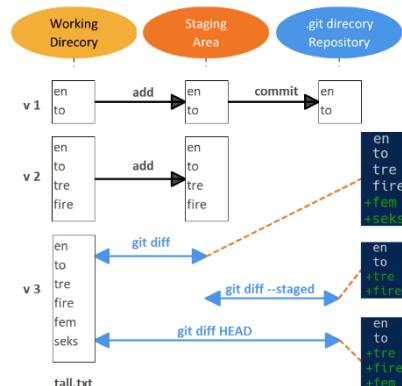
- `git config --global --list`

Kommandoene ovenfor utfører vi kun én gang. Når vi vil starte med versjonskontroll i en mappe, bruker vi `git init`. Denne kommandoen oppretter en skjult `.git`-mappe som vi ikke skal røre, og som holder styr på alle versjonene.



For å forstå Git bør vi kjenne til forskjellene mellom *Working Directory*, *Staging Area* og *Repository*. *Working Directory* er katalogen hvor vi jobber med filene. Vi bestemmer selv hvilke av disse filene Git skal følge med på. Det gjør vi med `git add` kommandoen, som legger filen til *Staging Area*. `git commit` lagrer en versjon av filene fra *Staging Area* til *Repository*. Vi gjentar denne totrinnsprosessen hver gang vi ønsker en versjon av filene lagret permanent. Vi kan slå kommandoene sammen med `git commit -am <melding>` (gjelder endrede, men ikke nye filer)

Underveis kan vi bruke `git log`, `git status` og `git diff` for å få en oversikt over hva vi har gjort og hva vi holder på med. Ønsker vi å hente tilbake filinnholdet som ligger i *Staging Area*, bruker vi `git restore <filnavn>`. Dersom vi ønsker å få tilbaketidligere versjoner som ligger lagret i *Repository*, bruker vi `git restore --source main~n <filnavn>`, hvor *n* er antall versjoner bakover.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

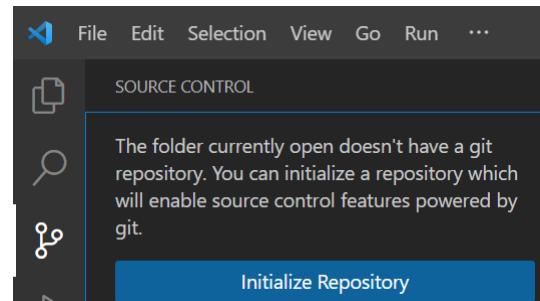
### Git-ordliste

Selv om vi i Norge sier skjerm i stedet for *screen*, bruker vi ofte *harddisk* og sjeldent platelager. Git bruker et engelsk kommandospråk, og vi vil holde oss mest mulig til de engelske ordene.

Engelsk uttrykk	Oversettelse	Forklaring
<i>repository</i>	<i>repo</i>	Et depot (kodelager) som sporer og lagrer historikken til alle endringer som er gjort i filene. Repositoryet ligger i den skjulte mappen <code>.git</code> .
<i>staging area</i>	<i>Staging-området</i>	Et midlertidig område hvor vi legger filer fra arbeidsområdet (ved å bruke <code>add</code> ) som skal lagres i <i>repo</i> -et (ved å bruke <code>commit</code> ).
<i>working directory</i>	Arbeidsområdet	Arbeidsområdet der vi jobber med filene. Her kan det også ligge filer som kun er lagret lokalt i denne mappen og ikke i <i>staging</i> -området eller <i>repo</i> -et. Filer som kun er lagret i arbeidsområdet kalles " <i>untracked files</i> ". Filer som ligger i <i>staging</i> -området kalles " <i>tracked files</i> ".
<code>git add</code>	adde	Legger til filer ( <i>tracked</i> ) fra arbeidsområdet inn i <i>staging</i> -området.
<code>git commit</code>	kommitte	Lagrer ( <i>tracked</i> ) filer fra <i>staging</i> -området inn i <i>repo</i> et.
<code>git restore</code>	restore	Henter inn filer fra <i>staging</i> -området eller <i>repo</i> -et til arbeidsområdet. <code>checkout</code> er en mer generell kommando som også brukes til dette formålet.
<i>branch</i>	gren	Versjonene lagres etter hverandre langs hovedlinjen ( <i>main branch</i> ). Når vi skal utvide funksjonaliteten i et program, er det vanlig å opprette en egen gren hvor vi også lagrer flere versjoner. Når utvidelsen er ferdig testet, fletter vi den sammen med hovedgrenen.
<code>git branch</code>	branche	Viser eller oppretter en gren ( <i>branch</i> )
<code>git switch</code>	switche	Går fra en gren til en annen. <code>checkout</code> er en mer generell kommando som også brukes til dette formålet.
<code>git merge</code>	merge	Fletter endringene fra en gren sammen med en annen gren.

### Git for personlig bruk med VS Code

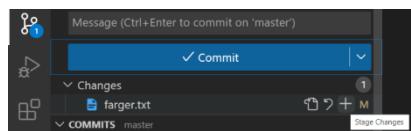
Først oppretter vi en mappe hvor vi ønsker å ha versjonskontroll, og så åpner vi den i VS Code. I verktøylinjen (*Activity Bar*) velger vi versjonskontroll (*Source Control*) og klikker på *Initialize Repository*.



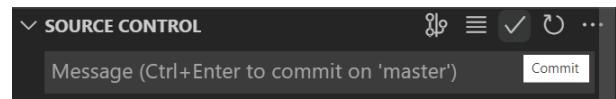
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

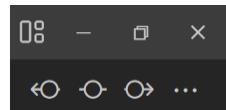
For å legge til endringer med `git add`, klikker vi på `+` tegnet (*Stage Changes*) ved siden av filnavnet.



Når vi ønsker å lagre en versjon med `git commit`, skriver vi først en melding som beskriver hvilke endringer som er gjort, og deretter klikker vi på avkryssingsknappen. Dette forklares godt i videoen [Using Git with Visual Studio Code](#).



Vi kan se hvilke endringer som er gjort siden forrige versjonslagring (*commit*) ved å klikke på filnavnet. For en bedre oversikt anbefaler vi utvidelsen **GitLens**. Med GitLens kan vi bla gjennom tidligere versjoner, sammenligne to versjoner, og mye mer. Vi kan til og med sammenligne to vilkårlige versjoner. Under *Commits* eller *File History (Primary Side Bar)* kan vi høyreklikke på en versjon av filen og velge *Select for Compare*. Deretter høyreklikker vi på den andre versjonen vi ønsker å sammenligne med, og velger *Compare with Selected*. En enda enklere måte er å trykke Alt og klikke på forrige-versjon-knappen. Da får vi en nedtrekksmeny med tidligere versjoner som vi kan sammenligne med.



Hvis vi vil gjenopprette en tidligere versjon, høyreklikker vi på den ønskede versjonen og velger *Restore (Checkout)*. Deretter kan vi lagre en ny versjon (av den gamle) med `git commit`, slik at vi beholder historikken. Det finnes også andre metoder for feilretting ([KM06](#)), for eksempel ved å bruke `git reset` og `git restore`. Vi kan finne mer informasjon om dette i [How to Undo Changes in Git](#).

Når vi skal legge til nye funksjoner i et program, er det vanlig å opprette sidegrenener (*branches*) i tillegg til hovedgrenen (*main*). I disse ekstra grenene, som ikke påvirker hovedgrenen, lagrer vi også flere versjoner. Når utvidelsen er ferdig testet, fletter vi den tilbake inn i hovedgrenen. Hvis det har blitt gjort endringer i hovedgrenen i mellomtiden, kan det oppstå konflikter som må løses. Mer om dette senere.

På nettet finner vi en rekke ressurser om Git, som videoer, veiledninger og ofte stilte spørsmål (FAQ), for eksempel [Git Reference](#), W3Schools [Git Tutorial](#), samt YouTube-videoene [Learn Git In 15 Minutes](#) og [Git and GitHub for Beginners - Crash Course](#).

### Ta med minst dette

**Git** er et verktøy for versjonskontroll som lar oss organisere og spore endringer i kode. Her bruker vi Git til personlig bruk, men Git kan også brukes i teamarbeid. Git kan brukes fra terminalvinduet, men det er enklere å bruke **VS Code** som har innebygd støtte for Git.

Utvidelsen **GitLens** gjør jobben enda enklere. Når vi skal bruke Git, må vi først opprette et *repo*, som lagres i en skjult mappe, `.git`, som vi ikke skal røre. Deretter må vi legge til (*adde*) filer fra arbeidsområdet til *staging*-området før de kan lagres permanent som en versjon i *repoet* med kommandoen *commit*. Det er verktøylinjer og menyvalg i VS Code for alt dette, samt å sammenligne versjoner, gå tilbake til gamle versjoner, opprette grener for å jobbe

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

parallelt, med mer. Kodingen blir sikrere fordi vi ikke mister tidligere kode når vi overskriver den. Det kan spare oss for mye bortkastet tid dersom det oppstår uventede problemer.

### Øvingsoppgaver

#### 2.3.1 Installer Git

Installer og konfigurer Git som beskrevet først i avsnittet [Git for personlig bruk fra kommandolinjen](#).

#### 2.3.2 Git med terminalvinduet

Hvis vi ønsker en visuell representasjon av Git-status i ledeteksten, kan vi først installere **Oh My Posh** i Windows eller **Oh My Zsh** i MacOS eller Linux, se [2.3.4 Terminaltilpasning med skreddersydde Git-promter](#). Utfør kommandoene nedenfor for å bli kjent med Git.

Kommandoer som kun gjelder for macOS og Linux og skiller seg fra Windows, er markert med **rød tekst**. Les og prøv å forstå kommandoenes tilbakemeldinger. I macOS bruker git ofte programmet **less** som en sideviser for terminalen. Det vil stå **less** på tittellinjen og : (kolon) eller **(END)** nederst i vinduet. Trykk **h** for hjelp, **u** for å bla opp, **d** for å bla ned eller **q** for å avslutte og returnere til terminalen.

Forklaring	Kommando	
1. Opprett en katalog kalt <b>repos</b> , for eksempel i hjemmekatalogen.	<code>cd</code> <code>md repos</code>	<code>cd</code> <code>mkdir repos</code>
2. Naviger til <b>repos</b> .	<code>cd repos</code>	
3. Initialiser et <i>repo</i> i katalogen <b>A</b> <sup>16</sup> .	<code>git init A</code>	
4. Naviger til A.	<code>cd A</code>	
5. Sjekk at <b>.git</b> -mappen er opprettet.	<code>dir -force</code>	<code>ls -la</code>
6. Sjekk <i>repo</i> -statusen.	<code>git status</code>	
7. Opprett filen <b>tall.txt</b> ved hjelp av <i>notepad</i> og skriv "en", "to", "tre" og "fire" på 4 linjer.	<code>notepad tall.txt</code>	<code>touch tall.txt</code> <code>open -e tall.txt</code>
8. Vis innholdet i filen <i>tall.txt</i>	<code>type tall.txt</code>	<code>cat tall.txt</code>
9. Sjekk <i>repo</i> -statusen.	<code>git status</code>	
10. Legg <i>tall.txt</i> til <i>staging</i> -området.	<code>git add tall.txt</code>	
11. Sjekk <i>repo</i> -statusen.	<code>git status</code>	
12. <i>Commit</i> endringene i <i>repo</i> -et.	<code>git commit -m "Første import av tall.txt"</code>	

<sup>16</sup>Dette oppretter samtidig A. Vi kan også opprette A, navigere til A og skrive `git init`.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

13. Sjekk repo-statusen.	git status	
14. Endre tall.txt slik at den inneholder 6 linjer med tallene fra 1 til 6.	notepad tall.txt	open -e tall.txt
15. Vis innholdet i tall.txt	type tall.txt	cat tall.txt
16. Vis forskjellen mellom tall.txt i arbeidsområdet og <i>staging</i> -området".	git diff	
17. Legg tall.txt til <i>staging area</i> .	git add tall.txt	
18. Sjekk repo-statusen og vis forskjellene mellom tall.txt i arbeidsområdet og <i>staging</i> -området, samt mellom arbeidsområdet og <i>repo</i> -et	git status git diff git diff HEAD	
19. Commit endringene i <i>repo</i> -et.	git commit -m "6 tall hvorav 2 nye"	
20. Vis <i>repo</i> -loggen.	git log	
21. Gjenopprett første versjon av tall.txt	git restore --source main~1 tall.txt	
22. Vis innholdet i filen.	type tall.txt	cat tall.txt
23. Commit endringene i <i>repo</i> -et. Ved å bruke <b>-am</b> hopper vi over <b>git add</b> -trinnet (gjelder ikke for nye filer)	git commit -am "restore 1. versjon med 4 tall"	
24. Vis tilgjengelige grener, opprett grenen <b>engelsk</b> , bytt til grenen <b>engelsk</b> og vis grener på nytt.	git branch git branch engelsk git switch engelsk git branch	
25. Vis innholdet i tall.txt. Endre innholdet slik at det igjen inneholder 6 tall, hvor de 3 siste er skrevet på engelsk.	type tall.txt notepad tall.txt type tall.txt	cat tall.txt open -e tall.txt cat tall.txt
26. Commit endringene i <i>repo</i> -et.	git commit -am "6 tall, de 3 siste på engelsk"	
27. Bytt til grenen <i>main</i> .	git switch main	
28. Vis innholdet i tall.txt	type tall.txt	cat tall.txt
29. Legg til teksten "fem" og vis innholdet i tall.txt	notepad tall.txt type tall.txt	open -e tall.txt cat tall.txt
30. Commit endringene i <i>repo</i> -et	git commit -am "5 tall på norsk"	
31. Merge innholdet fra grenen <b>engelsk</b> .	git merge engelsk	
32. Løs konflikten	notepad tall.txt	open -e tall.txt

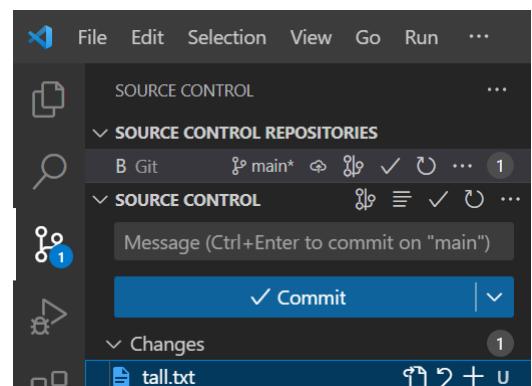
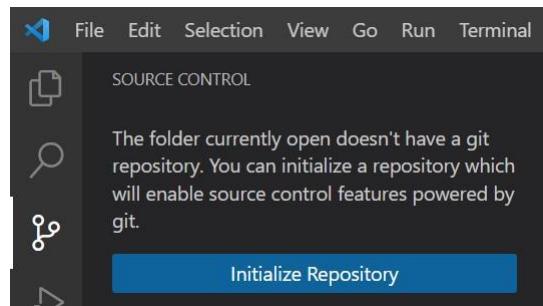
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

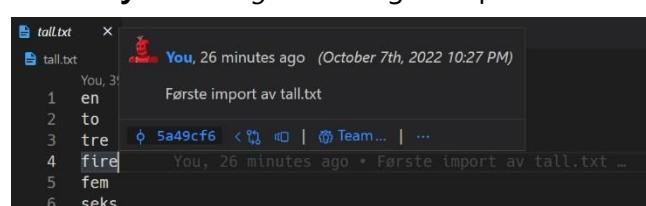
33. Commit endringene og prøv å merge innholdet på nytt.	git commit -am "merge branch engelsk" git merge engelsk
34. Vis en grafisk representasjon av historikken	git log --graph

### 2.3.3 Git med VS Code

1. Installer utvidelsen GitLens
2. I terminalvinduet (i VS Code), gå til **c:\repos** og opprett katalogen **B**
3. Åpne denne katalogen med File, Open Folder.
4. Velg *Source Control* fra verktøylinja til venstre og klikk på *Initialize Repository*
5. Velg Explorer fra verktøylinja og opprett filen tall.txt med 'en', 'to', 'tre' og 'fire' på 4 linjer.
6. Sjekk symbolene til høyre for filnavnet i *Source Control*.
7. Klikk på **+** tegnet for å tilføye filen til *Staging Area* (+U går til -A).
8. Lagre versjonen i repo-et: Skriv **Første import av tall.txt** i *Message*-feltet og klikk på *Commit*.
9. Sjekk *Commits* under *Source Control*. *Commit Details* og *File History* finnes under *GitLens Inspect*.
10. Endre tall.txt med to nye linjer med teksten 'fem' og 'seks'.
11. Klikk på tall.txt under *Source Control*



12. Bla deg fram og tilbake med GitLens knappene øverst til høyre
13. Klikk på **+** tegnet for å tilføye filen til *Staging Area*.
14. Lagre versjonen i repo-et. Skriv **6 tall hvorav 2 nye** i *Message*-feltet og klikk på *Commit*.
15. Lukk alle filene. Åpne tall.txt på nytt. Klikk på en linje og før muspekeren over den grå teksten.

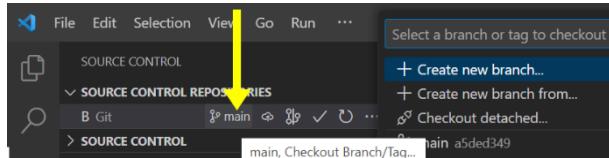


## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

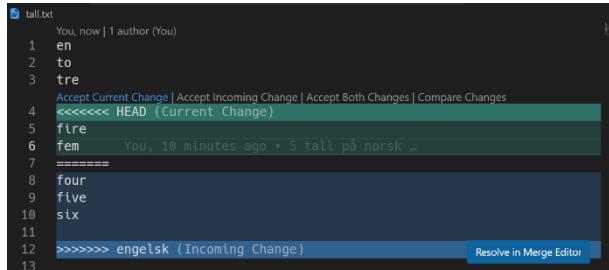
16. Sjekk *Commits*, *Commit Details* og *File History*.
17. *Restore* 1. versjon av tall.txt: Høyre-klikk på filen under *Commits* eller *File History* og velg *Restore (Checkout)*. Velg Explorer, åpne filen og sjekk at det bare er fire linjer.
18. Velg *Source Control* og lagre versjonen i repo-et: Skriv **restore 1. versjon med 4 tall** i *Message-feltet* og klikk på *Commit*.

19. Klikk på *main (checkout)*, under *Source Control Repositories* eller helt nede til venstre, og velg *Create new branch...* og skriv **engelsk**.

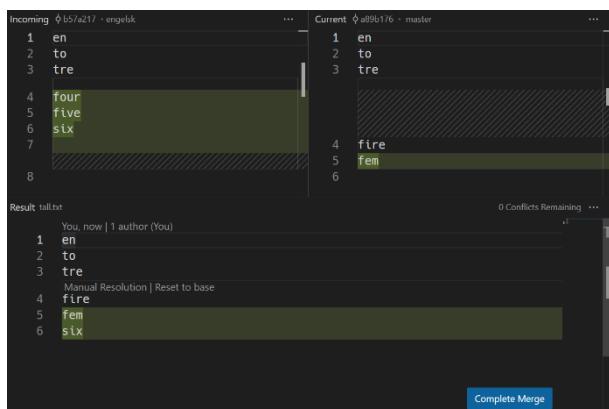


20. Sjekk at *Checkout*-knappen nå viser **engelsk** i stedet for **main**, og endre innholdet i tall.txt til **en, to, tre, four, five og six** på seks linjer.
21. Lagre versjonen: Skriv **6 tall, de 3 siste på engelsk** i *Message-feltet* og klikk på *Commit*.
22. Klikk på og *Checkout*-knappen velg **main**. Sjekk at tall.txt kun viser tallene fra en til fire.
23. Legg til teksten 'fem' på 5. linje, lagre filen, klikk på **+knappen** (add filen til *staging area*) og lagre versjonen: Skriv **5 tall på norsk** i *Message-feltet* og klikk på *Commit*.

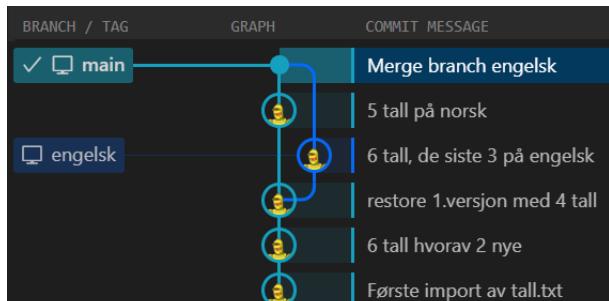
24. Under *Source Control*, klikk på de tre prikkene (*More actions*), velg *Branch*, *Merge Branch* og **engelsk**. Konflikten vises og kan løses her,



25. Eller vi kan klikke på *Resolve in Merge Editor* og gjøre det der. Til slutt klikker vi på *Complete Merge*.
26. Husk å lagre (*commit*) versjonen som er sammenflettet ('Merge branch engelsk').

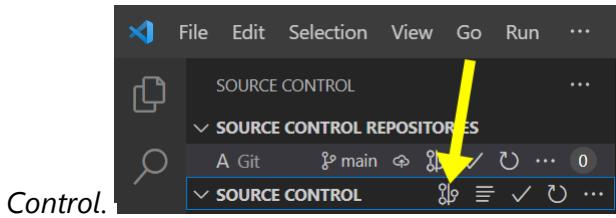


27. Klikk på *Show Commit Graph* under *Source Control Repositories* eller *Source*



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

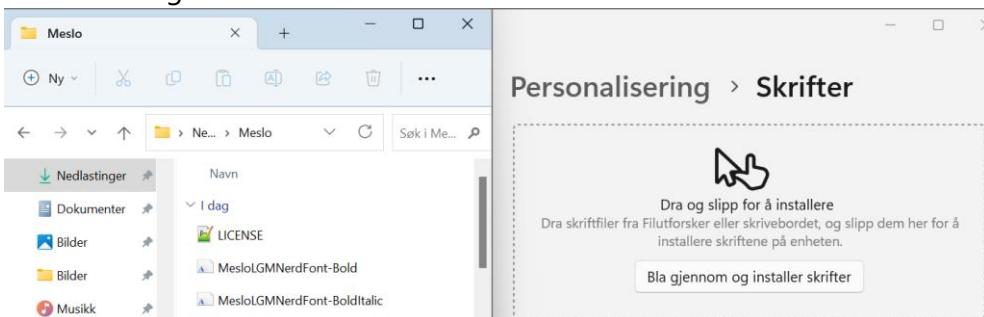


### 2.3.4 Terminaltilpasning med skreddersydde Git-prompt

#### Oh My Posh for Windows (Se også [Oh My Zsh](#) for MacOS og Linux)

Ledetekst, [cmd.exe](#), er stort sett en Windows-versjon av [command.com](#) i DOS, som vi brukte til å utføre operativsystemkommandoer på 1980-tallet. Vi opprettet kataloger, kopierte filer, stilte klokka, osv. fra kommandolinjen. Ledetekst (*prompt*) er teksten som vises i konsollvinduet (*Windows Console*) og betyr at programmet er klar til å ta imot kommandoer. PowerShell, [powershell.exe](#), er en mer avansert versjon av Ledetekst og kommer forhåndsinstallert med Windows som Windows PowerShell. Vi skal bruke en nyere versjon av PowerShell, [pwsh.exe](#), som kan kjøre på ulike plattformer. Dessuten kan vi erstatte *Windows Console* med *Windows Terminal* som lar oss bruke flere ulike kommandolinjer som [cmd](#), [powershell](#) og [bash](#) (Linux/Ubuntu) i hver sine arkaner.

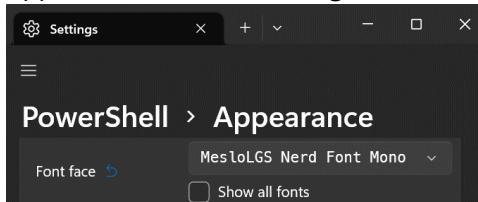
1. Installer siste versjon av [Windows Terminal](#) og [PowerShell](#) fra [Microsoft Store](#).
2. Start Terminal, Åpne Settings og velg *PowerShell* som standard.
3. Vi trenger noen ekstra symboler, så gå til [Nerd Fonts](#) og last ned [MesloLG Nerd Font](#), pakk ut zip-filen, åpne skrifter og dra filene over fra Meslo-mappen til Skrifter for å installere dem. Skrifter åpner vi ved å klikke på Windows knappen, skrive **skr** og velge skrifteinstillinger.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

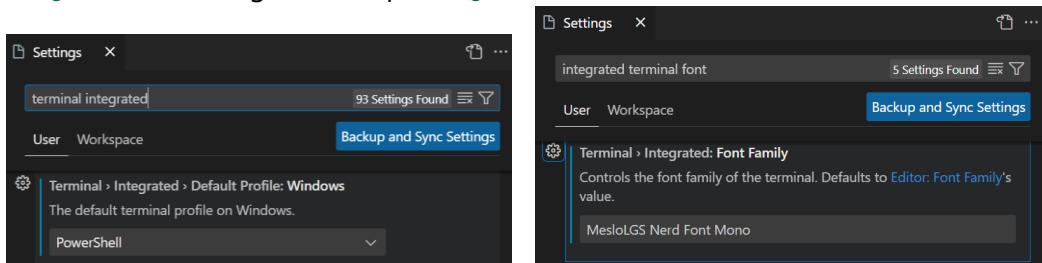
4. Start Terminal, åpne *Settings* og endre *Font face* til **Meslo LGS Nerd Font Mono** under *Appearance* for *Default* og *Powershell Profiles*.



5. Gå til [Oh My Posh](#), følg installasjonsveiledningen (winget), dvs lim inn denne kommandoen i PowerShell: `winget install JanDeDobbeleer.OhMyPosh -s winget`
6. Fortsett i PowerShell med å installere:
- ```
Install-Module -Name posh-git  
Install-Module -Name Terminal-Icons  
Install-Module -Name PSReadLine
```
7. Rediger profilen ved å skrive `notepad $PROFILE` og legg inn (se filen `b_2_3_profile.txt`)
- ```
$env:POSH_GIT_ENABLED = $true  
oh-my-posh init pwsh | Invoke-Expression  
Import-Module Terminal-Icons
```
8. Nå skal det komme fram git-informasjon i ledeteksten i en mappe med et git-repo.



9. For at dette også skal virke i *VS Code*, må vi velge *PowerShell* og *MesloLSG Nerd Font Mono*. Klikk på tannhjulet nede til venstre i vinduet, velg *Settings* og søk først på *default integrated terminal* og deretter på *integrated terminal font*.



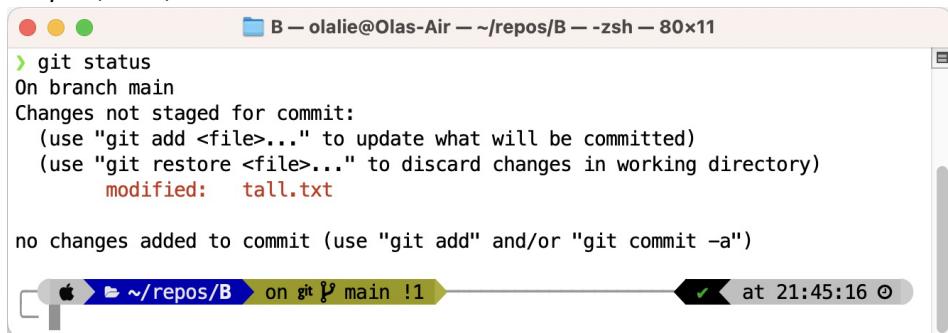
### Oh My Zsh for MacOS og Linux

I macOS kan vi [installere Oh My Zsh](#) med [curl](#), [4 Meslo skriftyper](#) og temaet [Powerlevel10k \(Manual, git clone...\)](#). Hvis konfigureringen av Powerlevel10 ikke starter automatisk, kan det igangsettes med `p10k configure`. Se for øvrig [Getting Started](#). Sjekk også at *Terminal*, *Settings*,

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

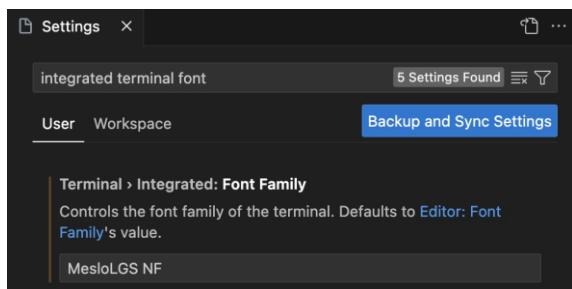
*Profiles, Text, Font* er satt til **MesloLGS NF**.



```
git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   tall.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

For at dette også skal virke i *VS Code*, kan standardterminalen for Osx stå til **null** (systeminnstillinger), men vi må sette *Terminal, Integrated, Font Family* til **MesloLGS NF**, se beskrivelsen rett ovenfor.



Vurderingsseksemplar

## 2.4 Funksjoner

### Bolkens innhold

Innledning .....	92
Bruk av parametere.....	93
Returverdier .....	95
Lokale og globale variabler .....	96
Overføring av argumenter .....	98
Flere emner innen funksjoner.....	99
Ta med minst dette.....	100
Øvingsoppgaver.....	100

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

### Innledning

Vi har allerede brukt mange funksjoner, både innebygde som `print()` og importerte som `mainloop()` i `tkinter` pakken. I 1.1 Grunnleggende programmering så vi hvordan vi kunne lage egne funksjoner, som f.eks. `finn_maks()`. Funksjoner er blokker med kode som vi kan utføre flere ganger ved å kalle opp funksjonen. Hvis vi gjentar samme kodelinjer flere steder i et program, tyder det på at vi bør legge disse linjene inn i en funksjon. Det er best å vurdere dette under planleggingen: Hvilke funksjoner kommer vi til å trenge? Finnes det allerede funksjoner vi kan benytte, eller må vi lage dem selv? Et program blir kortere og mer oversiktlig med funksjoner. Hvis det oppstår feil i disse kodelinjene, kan de rettes opp på ett sted. Dessuten kan vi lagre funksjoner i en modul (.py fil) som både vi og andre kan importere i andre programmer for å gjenbruke disse funksjonene ([KM07](#)).

En funksjon defineres med det reserverte ordet `def`. Så kommer **funksjonsnavnet**, som bør være i små bokstaver og forklarende. Deretter kommer vanlig **start- og sluttparentes** og til slutt **kolon**. Kodelinjene som følger må ha riktig innrykk, slik som i andre kodeblokker. Vi kaller opp funksjonen med funksjonsnavnet etterfulgt av start- og sluttparentes. Funksjonen må være definert før vi kan bruke den.

```
def hils():
    print('Hei, jeg er Eva.')

hils()
✓ 0.0s
Hei, jeg er Eva.
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Bruk av parametere

Hver gang vi kaller funksjonen `hils()` ovenfor, skriver den ut `Hei, jeg er Eva`. Vi kan gjøre den mer generell ved å angi ulike navn når vi kaller den opp ([KM07](#)). Da må funksjonen defineres med **parametere**. Det gjøres inni parentesene. Verdiene (`Alex` og `Beate`) vi bruker når vi kaller opp funksjonen, heter **argumenter**.

```
def hils(navn):
    print(f'Hei, jeg er {navn}.')

hils('Alex')
hils('Beate')
✓ 0.0s

Hei, jeg er Alex.
Hei, jeg er Beate.
```

Hvis vi glemmer å oppgi argumentet, får vi en feilmelding.

```
try:
    hils()
except TypeError as err:
    print(f'Feil: {str(err)}')
✓ 0.0s

Feil: hils() missing 1 required positional argument: 'navn'
```

Vi kan ha flere parametere, og de er posisjonsavhengige så lenge vi ikke navngir argumentene.

```
def hils(navn, bosted):
    print(f'Hei, jeg er {navn} og kommer fra {bosted}.')

hils('Elise', 'Bergen')
hils('Stavanger', 'Frida')
hils(bosted='Oslo', navn='Gustav')
✓ 0.0s

Hei, jeg er Elise og kommer fra Bergen.
Hei, jeg er Stavanger og kommer fra Frida.
Hei, jeg er Gustav og kommer fra Oslo.
```

Hvis vi gir parameterne standardverdier, kan argumentene utelates.

```
def hils(navn, bosted="ukjent"):
    if bosted == "ukjent":
        print(f"Hei, jeg er {navn}.")
    else:
        print(f"Hei, jeg er {navn} og kommer fra {bosted}.")

hils("Hans")
hils("Irene", "Tromsø")
✓ 0.0s

Hei, jeg er Hans.
Hei, jeg er Irene og kommer fra Tromsø.
```

I Python kan vi lage funksjoner som godtar et varierende antall argumenter. Stjerneoperatøren (\*) foran parameternavnet pakker de gjenværende argumentene inn i en tuppel ([tuple](#)), og dobbeltstjerneoperatøren (\*\*) foran parameternavnet pakker gjenværende, navngitte argumenter inn i en ordbok ([dict](#)). Dette gir funksjonene større fleksibilitet. Posisjonsbaserte

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

parametere må alltid angis foran parametere definert med \* eller \*\*, men vi kan ha navngitte argumenter (nøkkel-verdi-par) etter stjerneoperatørene. Det samme prinsippet lå til grunn i bok 2.1 om tupler og ordbøker. Der tildelte vi verdier til flere variabler samtidig fra en tuppel eller liste, og kunne samtidig tilordne gjenværende verdier til en tuppel om vi ønsket det.

```
def skriv_ut_fag(*fag):
    print(f'Datatype: {type(fag)}, lengde: {len(fag)}')
    if not fag:
        print("Jeg liker ingen fag.")
    else:
        tekst = ", ".join(fag)
        før, _, etter = tekst.rpartition(", ")
        print(f"Jeg liker {før} og {etter}.")\n\n
skriv_ut_fag()
skriv_ut_fag("engelsk", "tysk")
skriv_ut_fag("it", "biologi", "psykologi")
✓ 0.0s\n
Datatype: <class 'tuple'>, lengde: 0
Jeg liker ingen fag.
Datatype: <class 'tuple'>, lengde: 2
Jeg liker engelsk og tysk.
Datatype: <class 'tuple'>, lengde: 3
Jeg liker it, biologi og psykologi.
```

Her ønsker vi å lage en funksjon som skriver ut verdiene i tuppelen `fag`, kommaseparert samtidig som vi bytter ut det siste kommaet med " og ". Først konverterer vi tuppelen `fag` til en tekst ved å bruke metoden `, join(fag)`. Denne metoden kombinerer elementene i tuppelen ved å bruke komma og mellomrom som skilletegn. Deretter deler vi teksten i tre med `rpartition()` som returnerer en tuppel. Det første elementet er teksten før det vi søker etter, det andre er det vi søker etter, og det tredje er teksten etter det vi søker etter. "`r`" i `rpartition` betyr at vi skal søke fra høyre (slutten av teksten). Bruken av `_` (understrek) som variabelnavn er vanlig praksis når vi ønsker å ignorere, og ikke kommer til å bruke, en bestemt verdi. Til slutt setter vi det sammen i en f-streng med "og".

I det nesete eksempelet bruker vi et varierende antall nøkkel-verdi-par, angitt med \*\* foran parameternavnet, som dermed blir en ordbok (`dict`).

```
def skriv_ut_personalia(**personalia):
    print(f'Datatype: {type(personalia)}, lengde: {len(personalia)}')
    if not personalia:
        print("Ingen opplysninger.")
    else:
        for nøkkel, verdi in personalia.items():
            print(f"{nøkkel.title():10}: {str(verdi).title()}")
    print()\n\n
skriv_ut_personalia()
skriv_ut_personalia(navn="joakim jensen", bosted="jessheim")
skriv_ut_personalia(navn="karl kruse", født=1993, yrke="konsulent")
✓ 0.0s\n
Datatype: <class 'dict'>, lengde: 0
Ingen opplysninger.\n
Datatype: <class 'dict'>, lengde: 2
Navn      : Joakim Jensen
Bosted   : Jessheim\n
Datatype: <class 'dict'>, lengde: 3
Navn      : Karl Kruse
Født     : 1993
Yrke     : Konsulent
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Python er et dynamisk typet språk, noe som betyr at datatypen til en variabel bestemmes først når programmet kjøres. En variabel kan endre datatype basert på verdien den tilordnes. For eksempel kan en variabel først referere til en tekst og senere til et heltall. Dette gir fleksibilitet, men kan også føre til feil dersom vi ikke er forsiktige med hvordan vi bruker variablene. I språk som er statisk typet, for eksempel Java og C#, må datatypen til en variabel være definert i koden, altså før programmet kjøres. Da vil vi få feilmeldinger allerede når vi koder eller kompilerer (lager kjørbar versjon av koden) dersom vi prøver å tilordne en verdi av feil datatype til en variabel i koden vår.

Selv om Python ikke løser variabler til bestemte datatyper, gir *type hints* oss muligheten til å spesifisere hvilke datatyper funksjonene forventer i argumentene og gir i returverdier. For parametere angis datatypen etter et kolon (:), mens returverdien følger etter en pil (->). For eksempel, i `def min_funksjon(x: int) -> str:`, forteller vi at `x` skal være av type `int` og at funksjonen vil returnere en `str`. I VS Code vil disse hintene bli synlige når vi holder musepekeren over funksjonsnavnet i koden. Det forhindrer imidlertid ikke kjøretidsfeil dersom funksjonen mottar feil datatyper, men vi kan slå på *Type Checking Mode* for å bli advart om typefeil med en rød, bølget strek under argumentet mens vi koder.

### Returverdier

Foreløpig har vi sett på funksjoner som utfører en handling, som å skrive noe ut til skjermen. Funksjoner kan imidlertid også returnere verdier ved hjelp av `return`-kommandoen.

```
def f(x: float) -> float:
    return x**2 - 4*x + 3

print('f(x)=x^2-4x+3')
for x in range(6):
    print(f'f({x}) = {f(x)}')
✓ 0.0s

f(x)=x^2-4x+3
f(0) = 3
f(1) = 0
f(2) = -1
f(3) = 0
f(4) = 3
f(5) = 8
```

I denne sammenheng kan vi tenke på funksjoner som svarte bokser: vi forer dem med inndata, som de behandler, og så får vi resultatet som utdata. Her har vi en funksjon `f` som tar imot en verdi `x` og returnerer resultatet av andregradspolynomet  $x^2 - 4x + 3$

```
def kvadrat_og_kubikk(x: float) -> tuple[float, float]:
    return x**2, math.pow(x, 3)

print("x, x^2 og x^3")
for x in 1, 3, 6:
    x_2, x_3 = kvadrat_og_kubikk(x)
    print(f"x = {x:2}, x^2 = {x_2:2} og x^3 = {x_3:5}")
✓ 0.0s

x, x^2 og x^3
x = 1, x^2 = 1 og x^3 = 1.0
x = 3, x^2 = 9 og x^3 = 27.0
x = 6, x^2 = 36 og x^3 = 216.0
```

Siden vi kan returnere objekter, kan et funksjonskall returnere flere verdier. Her regner vi ut potenser på to forskjellige måter, med `**`- operatoren og med `math.pow`-metoden. Resultatet som returneres er en tuppel.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

I det neste eksempelet bruker vi funksjonen `vis_sirkel_info()`, som forventer en sirkels radius som argument (inndata). Funksjonen returnerer en ordbok med verdier for nøklene `radius`, `omkrets` og `areal` som utdata.

```
def vis_sirkel_info(radius: float) -> dict:
    omkrets = 2 * math.pi * radius
    areal = math.pi * radius**2
    return {"radius": radius, "omkrets": omkrets, "areal": areal}

for radius in 3, 4:
    print("\nSirkelinfo for radius", radius)
    for nøkkel, verdi in vis_sirkel_info(radius).items():
        print(f"{nøkkel.title():8}: {verdi:5.2f}")

✓ 0.0s

Sirkelinfo for radius 3
Radius   : 3.00
Omkrets : 18.85
Areal   : 28.27

Sirkelinfo for radius 4
Radius   : 4.00
Omkrets : 25.13
Areal   : 50.27
```

`for method in dir(math):` `math`-modulen inneholder mange flere funksjoner i tillegg til `pow`, eksempelvis trigonometriske- og logaritmiske funksjoner,

konstantene `pi` og `e`, med mer. Vi kan skrive `math`. i VS Code så vil IntelliSense vise metodene som vi kan bruke til å fullføre uttrykket. Vi kan slå opp i [dokumentasjonen](#), eller vi kan liste ut metodene selv med kode.

### Lokale og globale variabler

Hvis vi et øyeblikk ser bort fra objektorientert programmering, finnes det to hovedtyper variabler.

**Lokale variabler** er definert innenfor funksjoner og er kun tilgjengelige innenfor funksjonen der de er definert. Når funksjonen har kjørt ferdig, forsvinner disse variablene. På den annen side er **globale variabler** vanligvis definert utenfor funksjoner og kan nås fra hvor som helst i koden, både utenfor og innenfor alle funksjoner.

```
def eksempel_1():
    lokal_var = "Jeg er lokal."
    print(f'Inni funksjonen, lokal_var: {lokal_var}')
    print(f'Inni funksjonen, global_var: {global_var}')

    global_var = "Jeg er global."
    print(f'Utenfor, før funksjonskallet: {global_var}')
    eksempel_1()
    print(local_var)

⊗ ⚡ 0.0s

Utenfor, før funksjonskallet: Jeg er global.
Inni funksjonen, lokal_var: Jeg er lokal.
Inni funksjonen, global_var: Jeg er global.

-----
NameError: name 'local_var' is not defined
```

I vårt første eksempel har vi en global variabel, `global_var`, og en lokal variabel `lokal_var` inni `eksempel_1`-funksjonen. Den lokale variablene er kun tilgjengelig innenfor funksjonen, mens den globale variablene er tilgjengelig både utenfor og innenfor funksjonen. Hvis vi forsøker å benytte `local_var` utenfor funksjonen, vil VS Code markere navnet med en gul, bølget linje og gi en advarsel om at `name 'local_var' is not defined`. Om vi likevel

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

forsøker å kjøre programmet, vil det stoppe med unntaket **NameError**.

```
def eksempel_2():
    global_var = "Jeg er nå lokal."
    print(f'Inni funksjonen: {global_var}')

    global_var = "Jeg er global."
    print(f'Utenfor, før funksjonskallet: {global_var}')
    eksempel_2()
    print(f'Utenfor, etter funksjonskallet: {global_var}')

✓ 0.0s

Utenfor, før funksjonskallet: Jeg er global.
Inni funksjonen: Jeg er nå lokal.
Utenfor, etter funksjonskallet: Jeg er global.
```

Hvis vi prøver å tilordne en verdi til den globale variabelen inni funksjon, oppretter vi en ny lokal variabel med samme navn som "skygger" for den ekte globale variabelen, som forblir uendret. Vi anbefaler å bruke forskjellige navn på lokale og globale variabler for å unngå forvirring.

```
def eksempel_3():
    global global_var
    global_var = "Jeg har blitt endret."
    print(f"Inni funksjonen,: {global_var} ID: {id(global_var)}")

    global_var = "Jeg er global."
    print(f'Utenfor, før funksjonskallet: {global_var} ID: {id(global_var)}')
    eksempel_3()
    print(f'Utenfor, etter funksjonskallet: {global_var} ID: {id(global_var)}')

✓ 0.0s

Utenfor, før funksjonskallet: Jeg er global. ID: 1648458497648
Inni funksjonen,: Jeg har blitt endret. ID: 1648458248992
Utenfor, etter funksjonskallet: Jeg har blitt endret. ID: 1648458248992
```

Hvis vi ønsker å tilordne en verdi til den globale variabelen inni funksjon, sier vi først fra at variabelen er global med **global global\_var**. Da kan vi heller ikke ha en lokal variabel med samme navn. Legg merke til at den globale variabelen

ikke er endret, men er et nytt objekt med ny ID. Dette er fordi **str** er uforanderlig (*immutable*).

Selv om det ikke anbefales, er det mulig både å ha og nå globale og lokale variabler med samme navn. Det kan imidlertid være nyttig å kjenne til funksjonene **locals()** og **globals()**.

```
def eksempel_4():
    global_var = "Jeg er nå lokal."
    print(f"Inni funksjonen, global_var: {global_var} ID: {id(global_var)}")
    globals()["global_var"] = "Jeg har blitt endret."
    tekst = 'Inni funksjonen, globals()["global_var"]:'"
    print(f'{tekst} {globals()["global_var"]} ID: {id(globals()["global_var"])}')

    global_var = "Jeg er global."
    print(f'Utenfor, før funksjonskallet: {global_var} ID: {id(global_var)}')
    eksempel_4()
    print(f'Utenfor, etter funksjonskallet: {global_var} ID: {id(global_var)}')

✓ 0.0s

Utenfor, før funksjonskallet: Jeg er global. ID: 1648458531952
Inni funksjonen, global_var: Jeg er nå lokal. ID: 1648458180112
Inni funksjonen, globals()["global_var"]: Jeg har blitt endret. ID: 1648458246912
Utenfor, etter funksjonskallet: Jeg har blitt endret. ID: 1648458246912
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
rabatt_grense = 1000
rabatt_prosent = 20

def rabatt(pris: float) -> float:
    if pris > rabatt_grense:
        rabatt = (pris-rabatt_grense)*rabatt_prosent/100
    else:
        rabatt = 0
    return rabatt

pris = 2499
print(f'Pris: {pris:7.2f}, rabatt: {rabatt(pris):7.2f}')
rabatt_grense = 2000
print(f'Pris: {pris:7.2f}, rabatt: {rabatt(pris):7.2f}')
✓ 0.0s
Pris: 2499.00, rabatt: 299.80
Pris: 2499.00, rabatt: 99.80
```

Vi ønsker å begrense bruken av globale variabler fordi det kan medføre at funksjoner returnerer forskjellige verdier med samme argumenter. Her kaller vi funksjonen `rabatt` med samme pris to ganger, men funksjonen returnerer forskjellige verdier fordi "noen" (kan være en annen funksjon) har endret på den globale variabelen `rabatt_grense` i mellomtiden. Slike funksjoner tillater vi ikke i matematikken og heller ikke i funksjonell programering

(paradigme) som krever at funksjonene skal være "rene". RENE funksjoner gir alltid samme svar med samme argumenter, og de har heller ingen bivirkninger (endrer på noe utenfor sitt lokale miljø). Så det er normalt bedre å returnere resultatet fra en utregning i en funksjon enn å lagre resultatet i en global variabel som vi kan hente opp i andre funksjoner.

### Overføring av argumenter

Parameterne i en funksjon oppfører seg som lokale variabler. Argumentene overføres ved å sende **en kopi av referansen** til objektene. Dette betyr at selv om referansen kopieres, peker begge (den originale og den kopierte) i utgangspunktet på det samme objektet. Vi kan bruke `id`-funksjonen til å bekrefte dette.

```
def endre(b: int) -> None:
    print(f'inni, b={b}, id(b)={id(b)}')
    b = 8
    print(f'inni, b={b}, id(b)={id(b)}')

    a = 7
    print(f'utenfor, a={a}, id(a)={id(a)}')
    endre(a)
    print(f'utenfor, a={a}, id(a)={id(a)}')
✓ 0.0s

utenfor, a=7, id(a)=140708217021416
inni, b=7, id(b)=140708217021416
inni, b=8, id(b)=140708217021448
utenfor, a=7, id(a)=140708217021416
```

Noen datatyper i Python er uforanderlige (*immutable*), som `int`, `str`, `bool`, `float` og `tuple`. Her har vi global variabel `a = 7`. Når vi sender `a` som argument til funksjonen, peker både `a` og `b` på samme objekt `7`, og `id(a)` og `id(b)` vil være like. Siden `7` er uforanderlig, kan vi ikke endre denne verdien. Når vi utfører `b = 8` inni funksjonen, oppretter vi et nytt objekt `8` og lar `b` peke på det. `id(b)` vil nå være forskjellig fra `id(a)`, som fremdeles peker på det opprinnelige objektet `7`. Selv om det kan virke som om vi overfører en kopi av verdien, sender vi altså

med en kopi av referansen. Dette tilsvarer det som i andre språk blir kalt for verdioverføring eller *passed by value*. Hvis vi hadde gitt parameteren i funksjonen navnet `a` i stedet for `b`, ville vi ha hatt to variabler med samme navn: en lokal innenfor funksjonen og en global utenfor. Men den lokale variabelen ville ikke påvirke den globale, og den globale `id(a)` ville ha vært forskjellige fra den lokale `id(a)` om vi tilordnet den en ny verdi.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
def endre(b: list) -> None:
    print(f'inni, b={b}, id(b)={id(b)}')
    b.append('Prinsessa')
    print(f'inni, b={b}, id(b)={id(b)}')
    b = ['Per', 'Pål', 'Espen', 'Prinsessa', 'Kongen']
    print(f'inni, b={b}, id(b)={id(b)}')

a = ['Per', 'Pål', 'Espen']
print(f'utenfor, a={a}, id(a)={id(a)}')
endre(a)
print(f'utenfor, a={a}, id(a)={id(a)}')
✓ 0.0s

utenfor, a=['Per', 'Pål', 'Espen'], id(a)=2044188433984
inni, b=['Per', 'Pål', 'Espen'], id(b)=2044188433984
inni, b=['Per', 'Pål', 'Espen', 'Prinsessa'], id(b)=2044188433984
inni, b=['Per', 'Pål', 'Espen', 'Prinsessa', 'Kongen'], id(b)=2044188434624
utenfor, a=['Per', 'Pål', 'Espen', 'Prinsessa'], id(a)=2044188433984
```

Python har også foranderlige (*mutable*) datatyper, eksempelvis `list` og `dict`. Igjen refererer `a` og `b` til samme liste når vi kommer inn i funksjonen, og `id(a)` og `id(b)` er like fordi de peker på samme objekt. Dermed blir også `a` påvirket når vi endrer på listen ved å bruke `b.append('Prinsessa')`. Når vi deretter oppretter en ny

liste med `b = ['Per', 'Pål', 'Espen', 'Prinsessa', 'Kongen']`, peker ikke `b` lenger på den opprinnelige listen, og `id(b)` blir en annen. Men `a` forblir uendret for den peker fremdeles på den originale (globale) listen. Derfor kommer `Prinsessa` med i `a`, men `Kongen` gjør det ikke. Dette kalles for referanseoverføring *passed by reference*. Som vi så tidligere, er verdioverføring i Python også en referanseoverføring, noe som skyldes hvordan Python håndterer dynamisk tilordning.

### Flere emner innen funksjoner

Vi har nå lært om grunnleggende prinsipper som parametere, returverdier, *type hints*, lokale og globale variabler og overføring av argumenter. Det finnes imidlertid flere emner innen funksjoner som gjør dem enda mer anvendelige.

- **Anonyme funksjoner (Lambda):** Små funksjoner uten navn for enklere oppgaver, beskrevet i [1.9 Reelle datasett med CSV](#).
- **Dekoratører:** Modifiserer eller utvider funksjonaliteten til andre funksjoner eller metoder. Vi tar i bruk dekoratører allerede i neste bok [2.5 Programvaretesting](#).
- **Docstrings:** Beskrivelser inne i en funksjon som forklarer hva den gjør, noe vi skal lære om i bok [3.6 Dokumentasjon](#)
- **Rekursjon:** Funksjoner som kaller seg selv og løser oppgaver ved å bryte dem ned i mindre, lignende delproblemer. Se vedlegg [5.4 Rekursjon](#)
- **Feilhåndtering:** Hvordan vi bruker `try/except` direkte inni funksjonen, eller utenfor der hvor funksjonen kalles.
- **Generatorer:** Funksjoner som bruker `yield` for å produsere data fortløpende uten å lage hele sekvensen i minnet på en gang, en effektiv metode for store datamengder.
- **Closures:** Når en indre funksjon husker variabler fra en ytre funksjon selv etter at den ytre funksjonen har kjørt ferdig, for eksempel når den ytre funksjonen returnerer den indre. Dette lar oss tilpasse hvordan en funksjon virker.

Selv om vi dekker mye om funksjoner, så er det alltid mer å utforske.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

Funksjoner lar oss gjenbruke kodeblokker og forenkler feilsøking og vedlikehold av programmer. Før vi skriver en funksjon av et visst omfang eller kompleksitet, bør vi undersøke om det finnes biblioteker med ferdige funksjoner som gjør jobben for oss. En funksjon defineres med nøkkelordet `def`, etterfulgt av et funksjonsnavn og eventuelle parametere omgitt av parenteser, og avsluttes med en kolon, som i `def hils(navn):`. For å bruke en funksjon, kalles den opp med funksjonsnavnet og de nødvendige argumentene inni parenteser, som i `hils("Eva")`. Funksjoner kan også returnere resultater med nøkkelordet `return`. Parametere gjør funksjoner mer fleksible, og de fungerer som lokale variabler. Globale variabler er tilgjengelige overalt i koden, mens lokale variabler kun er tilgjengelige inni funksjonen og forsvinner når funksjonen avsluttet. Når lister og andre foranderlige datatyper overføres som argumenter, vil eventuelle endringer i disse forblive endret selv etter at funksjonen har kjørt ferdig. Bolken avsluttet med en kort omtale av mer avanserte funksjonsemner.

### Øvingsoppgaver

#### 2.4.1

Lag en funksjon som tar inn to tall og returnerer summen, differansen, produktet og kvotienten av de to tallene som en tuppel. I denne oppgaven kreves ikke feilhåndtering dersom det andre tallet er null.

```
a, b = 2, 3
sum_, diff, prod, kvot = beregninger(a, b)
print(f'a: {a}, b: {b}')
print(f'Sum: {sum_}, Diff: {diff}, Prod: {prod}, Kvot: {kvot:.2f}')

✓ 0.0s

a: 2, b: 3
Sum: 5, Diff: -1, Prod: 6, Kvot: 0.67
```

#### 2.4.2

Lag en funksjon som fjerner alle tall over 10 fra en liste med tall.

Funksjonen skal returnere en ny liste.

```
tall = [4, 12, 3, 8, 23, 5, 11, 9]
print(fjern_over_10(tall))
print(tall)

✓ 0.0s

[4, 3, 8, 5, 9]
[4, 12, 3, 8, 23, 5, 11, 9]
```

#### 2.4.3

Lag en funksjon som skriver ut hvor mange vokaler det er i en tekststreng.

```
tekst = "Vil du være med på turen i høst?"
print(tekst)
print(f'Antall vokaler: {antall_vokaler(tekst)}')

✓ 0.0s

Vil du være med på turen i høst?
Antall vokaler: 10
```

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 2.4.4

Lag en funksjon som skriver ut hvor mange forskjellige bokstaver det er i en tekststreng. Når vi teller, skal vi ikke skille mellom store og små bokstaver. Hint: Du kan sjekke om en karakter er en bokstav med `karakter.isalpha()`.

```
tekst = "Sesam, sesam, lukk deg opp!"  
print(tekst)  
print(f'Antall unike bokstaver: {unique_bokstaver(tekst)}')  
  
✓ 0.0s  
  
Sesam, sesam, lukk deg opp!  
Antall unike bokstaver: 11
```

### 2.4.5

- Lag en funksjon som skriver ut en pyramide som vist nedenfor til venstre. Prøv å gjøre det med en `for`-løkke med bare én setning. Tips: `4**" "` er " ".
- Lag en funksjon som skriver ut en liten (S), medium (M) eller stor (L) pyramide. Funksjonen skal altså defineres med en parameter og kalles opp med et argument. Tillatt både store og små bokstaver. Se de to bildene under til høyre

```
pyramide()  
✓ 0.0s  
  
*  
***  
*****  
*****  
*****  
*****  
  
print('Liten pyramide:')  
pyramide('s')  
print('\nMedium pyramide:')  
pyramide('m')  
print('\nStor pyramide:')  
pyramide('L')  
  
✓ 0.0s  
  
Liten pyramide:  
*  
***  
*****  
  
Medium pyramide:  
*  
***  
*****  
*****  
*****  
  
Stor pyramide:  
*  
***  
*****  
*****  
*****  
*****  
*****  
*****
```

### 2.4.6

Anta det er 50 km mellom Askim og Oslo. Det er kø, og vi har en gjennomsnittshastighet på 50 km/t til Oslo. Hjem igjen er det ingen trafikk på motorveien og gjennomsnittshastigheten er 100 km/t. Gjennomsnittshastigheten tur/retur Oslo er ikke 75 km/t (aritmetisk gjennomsnitt), men 67 km/t (harmonisk gjennomsnitt). Det innser vi lett når vi har kjørt 100 km på 1,5 timer. Formelen for harmonisk gjennomsnitt er

$$\frac{n}{\left(\frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3} + \dots + \frac{1}{x_n}\right)}$$

Vurderingsseksempler

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Lag en funksjon, `harm_gjsn`, som tar som argument en liste med verdier og returnerer det harmoniske gjennomsnittet av verdiene.

```
lst = [50, 100]
print(f'Det harmoniske gjennomsnittet til {lst} er {harm_gjsn(lst):.1f}')\n\nlst = [5, 15, 25]
print(f'Det harmoniske gjennomsnittet til {lst} er {harm_gjsn(lst):.1f}')\n✓ 0.0s\n\nDet harmoniske gjennomsnittet til [50, 100] er 66.7
Det harmoniske gjennomsnittet til [5, 15, 25] er 9.8
```

#### 2.4.7

Lag en funksjon, `tall_med_siffer`, som tar som argument et siffer (0-9), og returnerer en liste med alle tall fra 0 til 99 som inneholder det angitte sifferet.

```
siffer = 5
print(f'Heiltall mellom 0 og 99 som inneholder sifferet {siffer}:')
print(tall_med_siffer(siffer))\n\nsiffer = 7
print(f'\nHeiltall mellom 0 og 99 som inneholder sifferet {siffer}:')
print(tall_med_siffer(siffer))
✓ 0.0s\n\nHeiltall mellom 0 og 99 som inneholder sifferet 5:
[5, 15, 25, 35, 45, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 65, 75, 85, 95]\n\nHeiltall mellom 0 og 99 som inneholder sifferet 7:
[7, 17, 27, 37, 47, 57, 67, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 87, 97]
```

Vurderingsseksempler

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 2.5 Programvaretesting

### Bolkens innhold

Innledning .....	103
pytest.....	104
Testdrevet utvikling .....	106
Parametere.....	107
Markeringer .....	108
Avanserte testteknikker.....	109
Ta med minst dette.....	109
Øvingsoppgaver.....	109

### Kompetansemål:

- vurdere og bruke strategier for feilsøking og testing av programkode

### Innledning

Når vi utvikler programvare, er målet å produsere kode uten feil som fungerer som forventet. Men det er lettere sagt enn gjort. Derfor tester vi programmene før vi tar dem i bruk. Som nevnt i [1.8 Modellering](#), kan testing utgjøre opp til 50% av tidsforbruket i systemutviklingsprosessen. Det finnes flere forskjellige tester som utføres på ulike nivåer. De vanligste er

1. **Enhetstesting:** Først tester vi hver enkelt enhet, som en funksjon eller klasse, for seg selv.
2. **Integrasjonstesting:** Deretter sjekker vi om disse enhetene fungerer riktig sammen med andre deler i systemet.
3. **Systemtesting:** Neste steg er å teste hele systemet for å se om det tilfredsstiller alle kravene.
4. **Akseptansetesting:** Nå er det brukernes tur til å teste programmet som om det allerede var satt i drift. Målet er å forsikre seg om at systemet fungerer som forventet og oppfyller kravspesifikasjonen. Det er ikke alltid utviklere og brukere er enige om hva spesifikke krav betyr, så det er viktig å snakke sammen underveis.

Som sagt finnes det flere typer tester, som eksempelvis sjekker ytelse, sikkerhet og kompatibilitet, med mer.

Vi bør ha en strukturert plan for hvordan vi vil teste, og jo mer som kan automatiseres desto bedre. Enhetstesting forteller oss å unngå vilkårlig testing, mens testing av grafiske brukergrensesnitt kan by på ekstra utfordringer. Foreløpige skal vi imidlertid fokusere på

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

enhetstesting og styre unna GUI. Enhetstesting ble utviklet på slutten av 1990-tallet, og først ute var Kent Beck and Erich Gamma med [JUnit](#) for Java. Python kommer med [unittest](#) som ligner på JUnit med testtilfeller (*test cases*) og testsamlinger (*test suites*). Vi skal bruke [pytest](#), som installeres enkelt i VS Code (`pip install -U pytest`), er gratis og åpen kildekode, lettere å lære, trenger mindre kode (slipper å opprette klasser) og har mer detaljert utskrift.

### pytest

La oss følge eksempelet i [Python testing in Visual Studio Code](#), hvor vi gjør det enkelt og har programmet og testprogrammet i samme mappe<sup>17</sup>.

1. Lag en mappe, `demo_tests`, og åpne denne mappen i VS Code.
2. Opprett filen `inc_dec.py` med innhold som vist til høyre.

```
def increment(x):
    return x + 1

def decrement(x):
    return x - 1
```

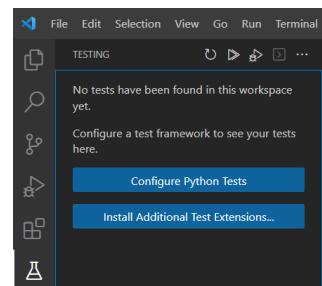
3. Opprett deretter filen `test_inc_dec.py` med innholdet som vist til høyre. Filene (modulene) og testene (funksjonene) må starte med `test_`.

```
import inc_dec

def test_increment():
    assert inc_dec.increment(3) == 4

def test_decrement():
    assert inc_dec.decrement(3) == 4
```

4. Klikk på begerglasset (Erlenmeyerkolben) i verktøylinja til venstre og deretter på *Configure Python Tests*. Vælg **pytest** (*Select a test framework/tool to enable*) og deretter **.Root Directory** (*Select the directory containing the tests*). Følg med i terminalvinduet hvor pytest installeres automatisk.



5. Vi kan nå kjøre alle testene fra terminalvinduet ved å skrive `pytest`,

```
platform win32 -- Python 3.10.6, pytest-7.2.0, pluggy-1.0.0
rootdir: C:\Users\olali\OneDrive - Viken fylkeskommune\Dokumenter\_IT-2\kompendium\eksempler\2_05_programvaretesting\2_05_1_demo_tests
plugins: anyio-3.6.2, dash-2.8.1, assert-utils-0.3.1, html-3.2.0, metadata-2.0.4
collected 2 items

test_inc_dec.py .F

===== FAILURES =====
test_decrement
=====
def test_decrement():
>     assert inc_dec.decrement(3) == 4
E     assert 2 == 4
E       + where 2 == <function decrement at 0x000001FDF1EE3520>(3)
E       +   where <function decrement at 0x000001FDF1EE3520> = inc_dec.decrement

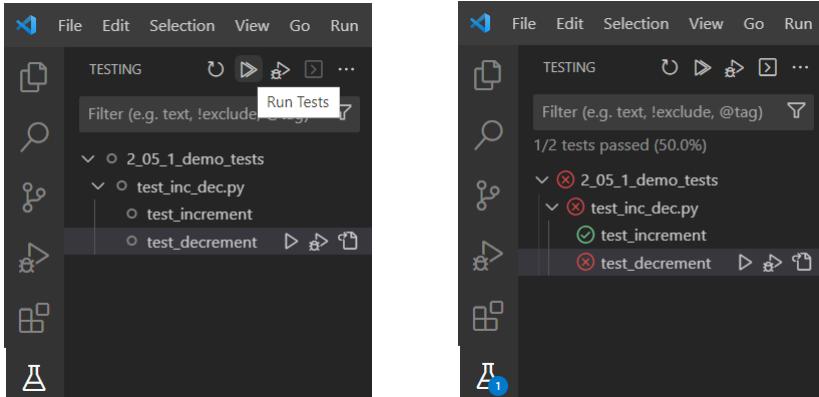
test_inc_dec.py:7: AssertionError
===== short test summary info =====
FAILED test_inc_dec.py::test_decrement - assert 2 == 4
===== 1 failed, 1 passed in 0.38s =====
```

<sup>17</sup> Se <https://docs.pytest.org/en/7.1.x/explanation/goodpractices.html>

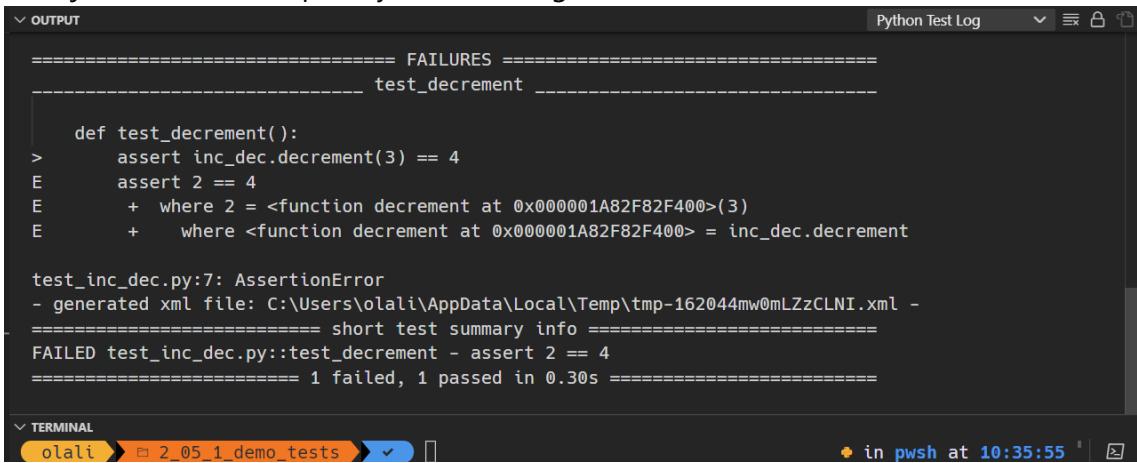
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

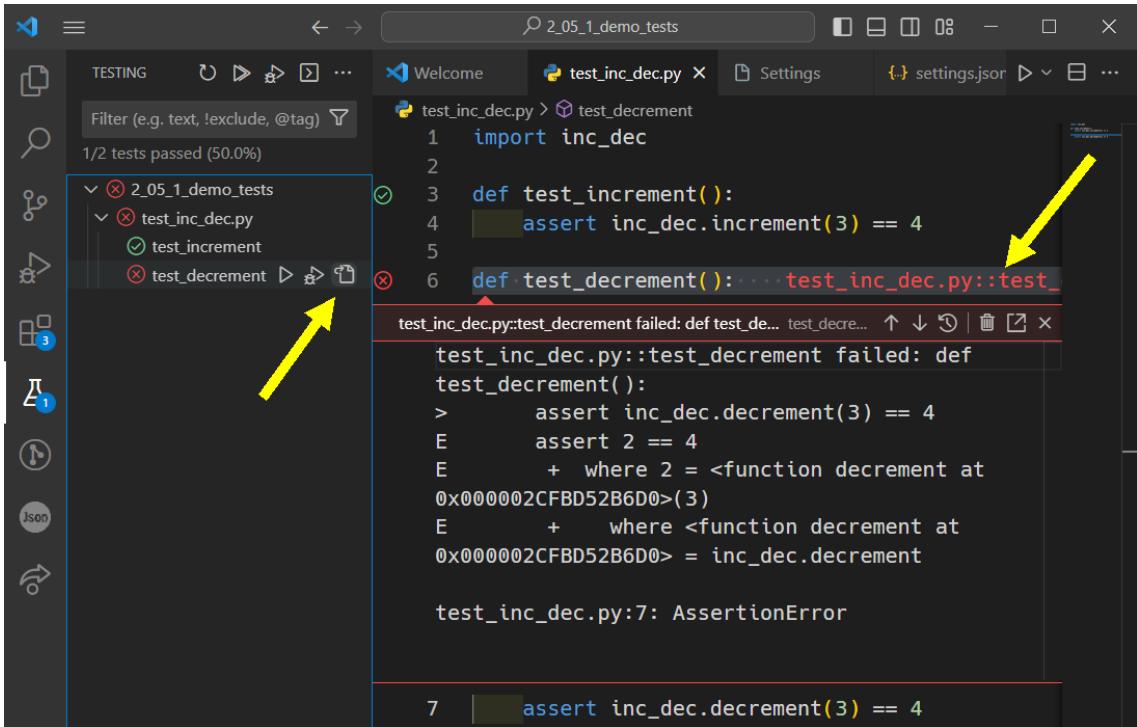
6. eller kjøre alle tester eller velge spesifikke tester i VS Code:



7. Detaljene kan ses i Output: Python Test Log,



8. eller bruke **Go to Test**-funksjonen og plassere musepekeren over koden, og eventuelt klikke på den



Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

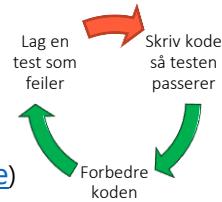
9. I Explorer-panelet i Visual Studio Code kan vi se at det har blitt opprettet tre nye undermapper: `__pycache__`, `.pytest_cache` og `.vscode`. Det er trygt å ignorere disse mappene og la **VS Code** og **pytest** håndtere dem automatisk.

### Testdrevet utvikling

Testdrevet utvikling (*TDD – Test-driven development*) innebærer å skrive testene først (som feiler), deretter skrive koden (så testene passerer) og til slutt omstrukturere koden (så den blir bedre). Når vi forbedrer (refaktorerer) koden, prøver vi å gjøre den enklere, og lettere å lese og vedlikeholde, uten å endre på hva den gjør. Testdrevet utvikling er godt egnet i sammenheng med smidige (agile) metoder. En vanlig tilnærming for å utføre enhetstester er å følge et *Arrange-Act-Assert*-mønster. Først setter vi opp testmiljøet, deretter utfører vi handlingen som skal testes, og til slutt sjekker vi at vi får det forventede resultatet.

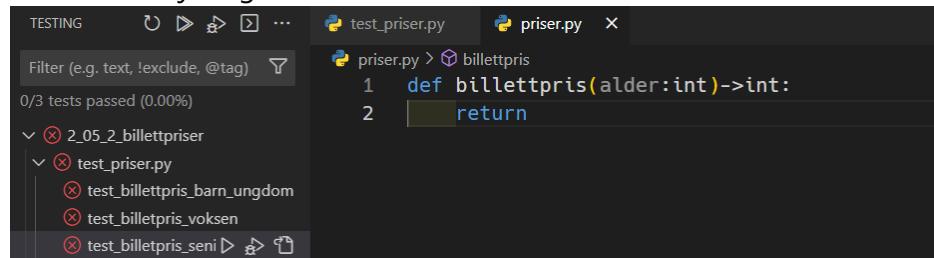
La oss ta en enkel funksjon,  
`billettpris(alder:int) -> int`, som skal returnere prisen i kroner for reisende i kategoriene **Barn/Ungdom**, **Voksen** eller **Honnør**, avhengig av alderen som er angitt i tabellen nedenfor. Anta at alderen allerede er kontrollert til å være et heltall mellom 6 og 130. Vi starter med å lage testene og en "tom" funksjon. Merk at vi tester en verdi midt i intervallet og også grenseverdiene.

Reisende	Alder	Pris
Barn/Ungdom	6-17 år	63
Voksen	18-66 år	157
Honnør	67 år eller mer	79



```
test_priser.py
test_priser.py > test_billettpris_senior
1 import priser
2
3 def test_billettpris_barn_ungdom():
4     assert priser.billettpris(6) == 63
5     assert priser.billettpris(10) == 63
6     assert priser.billettpris(17) == 63
7
8 def test_billettpris_voksen():
9     assert priser.billettpris(18) == 157
10    assert priser.billettpris(50) == 157
11    assert priser.billettpris(66) == 157
12
13 def test_billettpris_senior():
14     assert priser.billettpris(67) == 79
15     assert priser.billettpris(80) == 79
16     assert priser.billettpris(130) == 79
```

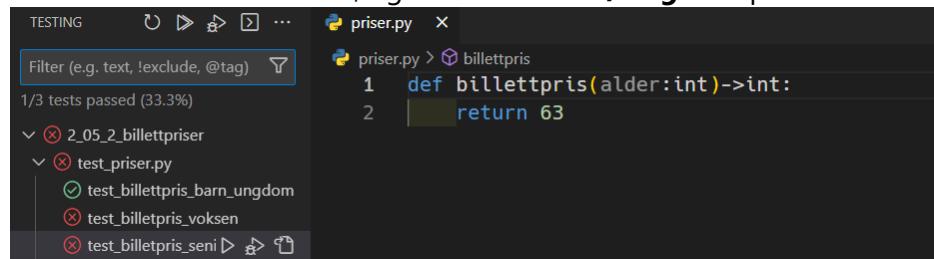
I den første kjøringen vil alle testene feile.



TESTING    Filter (e.g. text, !exclude, @tag)    test\_priser.py    priser.py  
0/3 tests passed (0.00%)  
  ↳ 2\_05\_2\_billettpriser  
    ↳ test\_priser.py  
      ↳ test\_billettpris\_barn\_ungdom  
      ↳ test\_billettpris\_voksen  
      ↳ test\_billettpris\_seni

```
priser.py > billettpris
1 def billettpris(alder:int)->int:
2     return
```

Vi endrer `return` til `return 63`, og testen for **Barn/Ungdom** passerer.



TESTING    Filter (e.g. text, !exclude, @tag)    priser.py  
1/3 tests passed (33.3%)  
  ↳ 2\_05\_2\_billettpriser  
    ↳ test\_priser.py  
      ↳ test\_billettpris\_barn\_ungdom  
      ✓ test\_billettpris\_voksen  
      ↳ test\_billettpris\_seni

```
priser.py > billettpris
1 def billettpris(alder:int)->int:
2     return 63
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Deretter legger vi til en if-setning: `if 17 < alder < 67: return 157`.

Dette gjør at testene for **Barn/Ungdom** og **Voksen** passerer.

The screenshot shows a PyCharm interface with two panes. The left pane displays a terminal window with test results: "2/3 tests passed (66.7%)". It lists three test files: "2\_05\_2\_billetpriser", "test\_priser.py", and "test\_billetpris\_voksen". The right pane shows the Python file "priser.py" with the following code:

```
def billettpris(alder:int)->int:
    if 17 < alder < 67: return 157
    return 63
```

Vi inkluderer deretter en `if`-blokk for Honnør-kategorien, og alle testene passerer.

The screenshot shows a PyCharm interface with two panes. The left pane displays a terminal window with test results: "3/3 tests passed (100%)". It lists four test files: "2\_05\_2\_billetpriser", "test\_priser.py", "test\_billetpris\_barn\_ungdom", "test\_billetpris\_voksen", and "test\_billetpris\_seior". The right pane shows the Python file "priser.py" with the following code:

```
def billettpris(alder:int)->int:
    if alder > 66: return 79
    if 17 < alder < 67: return 157
    return 63
```

Hvordan kan vi forbedre koden? I denne oppgaven, hvis én betingelse blir oppfylt, trenger ikke neste `if`-setning å bli utført, da

vi allerede har funnet aldersgruppen. Derfor er det bedre å bruke en `if-elif-else`-blokk. Vi kan også ha valgsettningene i en logisk rekkefølge og legge til kommentarer for å forbedre lesbarheten. Nedre grenseverdi trenger ikke å inkluderes i den første testen, siden vi har antatt at alderen allerede er kontrollert for å være mellom 6 og 130. Nedre grenseverdi trenger heller ikke å inkluderes i den andre testen, siden hvis vi kommer hit, betyr det at alderen er større enn 17. `if`- og `elif`-setningene vil da kunne utføres raskere, men her er det neppe tidskritisk, og vi beholder det som det er. Nå er aldersintervallene tydelige og kan enkelt sjekkes med tabellen. Hvis det hadde vært tidskritisk, kunne vi også vurdert å prioritere testene etter sannsynligheten for at betingelsen blir oppfylt. Hvorfor?

```
def billettpris(alder:int)->int:
    if 6 <= alder <= 17: return 63 # Barn/Ungdom
    elif 18 <= alder <= 66: return 157 # Voksen
    else: return 79 # Honnør
```

Reisende	Alder	Pris
Barn/Ungdom	6-17 år	63
Voksen	18-66 år	157
Honnør	67 år eller mer	79

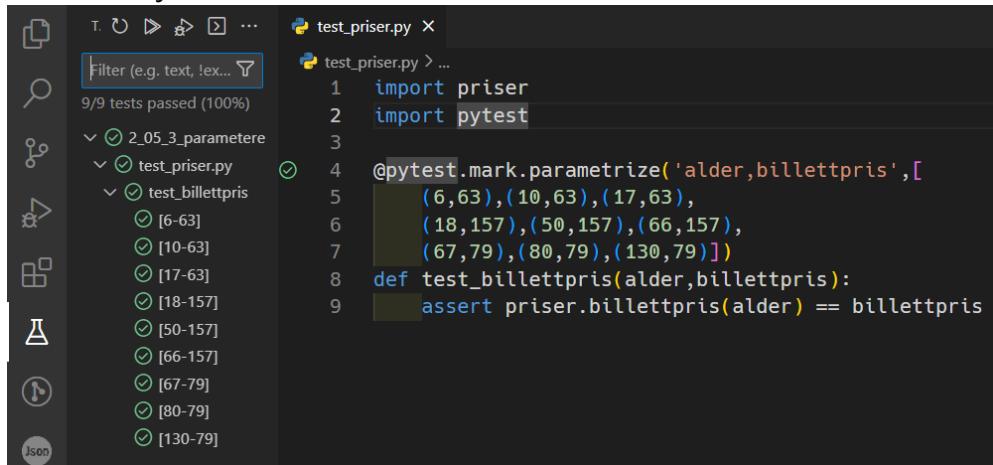
### Parametere

Vi kan gjøre testene våre mer effektive ved å 'parameterisere' dem. En enkelt test kan deretter kjøres flere ganger med ulike data. Ved å inkludere `@pytest.mark.parametrize`-dekoratøren over testfunksjonen og definere navnene på parameterne, som for eksempel `alder` og `billettpri`, sammen med en liste over argumenter i form av tupler, som `[(6, 63), (10, 63), ...]`, kan vi enkelt kjøre samme test med ulike kombinasjoner av verdier. Den første parameteren, `alder`, vil få tildelt de gitte aldersverdiene under testen, og den andre parameteren, `billettpri`, representerer det forventede resultatet som vi sammenligner med returverdien fra testen. Parameterne i test-funksjonen brukes som lokale variabler, likt som i

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

andre funksjoner.



```
test_priser.py
1 import priser
2 import pytest
3
4 @pytest.mark.parametrize('alder,billettpri', [
5     (6,63),(10,63),(17,63),
6     (18,157),(50,157),(66,157),
7     (67,79),(80,79),(130,79)])
8 def test_billettpri(alder,billettpri):
9     assert priser.billettpri(alder) == billettpri
```

Oppgave 6 på [IT-2 Eksamens Vår 2023](#) innebar å implementere fungerende kode basert på en angitt pseudokode<sup>18</sup> og utføre tester i henhold til en forhåndsdefinert testplan. Det finnes flere løsningsstrategier. Bruk av `print`-setninger, som vi allerede brukte da vi [installerte VS Code](#), gir rask tilbakemelding. Dette er en enkel tilnærming, men den krever manuell inspeksjon av hver test, noe som kan være tidkrevende og mindre effektivt ved omfattende tester. Med `assert`-setninger, som ble introdusert i bolken [1.7 Feil](#), stopper programmet ved første feil, noe som gjør at vi bare trenger å sjekke testene som feiler. Men en begrensning med `assert`-setninger er at hvis testen feiler, vil vi kun se en `AssertionError`-melding uten informasjon om hva funksjonen faktisk returnerte og hvorfor testen feilet. For å få denne informasjonen og svare på oppgaven, kan vi erstatte `assert`-setningen med en `print`-setning som viser resultatet av funksjonen. Hvis mange tester feiler, kan dette raskt bli uoversiktlig og tungvint. Et tredje alternativ er å skrive et dedikert testprogram basert på den angitte testplanen. Dette vil kjøre alle testene uten avbrudd og gi en oversikt over hvilke tester som passerer og feiler, samtidig som vi kan se resultatet funksjonen returnerer for testene som feiler. Selv om bruk av `pytest` med parameterer kan virke mer avansert, tar det ikke nødvendigvis lengre tid sammenlignet med enklere metoder som `print`- eller `assert`-setninger. I øving 2.5.2 blir det presentert en fungerende kode basert på den angitte pseudokoden. Oppgaven går ut på å utforske og sammenligne de tre alternative tilnærmingene for testing. Dette kan gi verdifull erfaring for valg av metode i ulike sammenhenger, selv når tiden er begrenset.

### Markeringer

`Parameterize` er en innebygd markering. Vi kan `markere` tester for å kategorisere dem og kontrollere hvilke tester som skal kjøres. Hvis vi skriver `@pytest.mark.linux` foran noen tester og deretter kjører `pytest -m linux`, vil kun disse testene bli kjørt. Foruten å navngi gruppene selv, finnes det flere innebygde markeringer. Her er tre eksempler.

1. `@pytest.mark.skip(reason="...")`  
**skip:** Testfunksjonen vi ikke bli kjørt

<sup>18</sup> Pseudokode er en generell beskrivelse av en algoritme, skrevet med et mer naturlig språk enn programmeringsspråk. Se [2.7 Noen enkle verktøy for systemutvikling](#).

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

2. `@pytest.mark.skipif(sys.version_info < (3, 10), reason="requires python3.10 or higher")`  
**skipif:** Testfunksjonen vil bare bli kjørt på bestemte vilkår
3. `@pytest.mark.xfail`  
**xfail:** Testen passerer dersom den lager en forventet feil/unntak (*exception*)

### Avanserte testteknikker

Senere skal vi også lære hvordan vi kan teste unntak (*exceptions*), bruk av *fixtures* og *mocking*<sup>19</sup>. *Fixtures* forbereder testtilfellene med felles kode for å unngå å duplisere. Dette kan inkludere ressurser som database tilkoblinger, variabler og objekter som testene trenger for å kunne kjøre. I andre enhetstestingverktøy kalles dette også for *setup/teardown*-, eller *beforeeach/aftereach*-metoder. *Mocking* lar oss simulere ('*mocke*') objekter, slik at vi kan teste funksjoner som er avhengige av eksterne data. For eksempel kan vi etterligne et tilfeldig terningkast ved å *mocke* biblioteket *random* eller simulere et API-kall som henter data fra en nettbasert database. Vi skal bruke *pytest-mock* for å få tilgang til *mocker-fixturen*, som gir oss *MagicMock* og *patch* for å enkelt kunne lage *mock*-objekter og erstatte funksjoner i testene våre.

### Ta med minst dette

Testing kan utgjøre opptil 50% av tidsforbruket i systemutviklingsprosessen. De vanligste testtypene er enhetstesting, integrasjonstesting, systemtesting og akseptanse testing. Vi bruker **pytest**, integrert i Visual Studio Code, til å kjøre testene. Filnavnene (modulene) og funksjonsnavnene (testene) må starte med **test\_**. Testprogrammene inneholder **assert**-setninger som sammenligner forventet verdi med virkelig verdi. Vi lagrer programmet og tilhørende testprogrammer i samme mappe, og konfigurerer testingen ved å klikke på begerglasset i verktøylinjen. Testene kan kjøres i terminalvinduet med **pytest**, og vi kan velge hvilke tester som skal kjøres ved å kategorisere dem med `@pytest.mark...`, men det er enklere å velge tester som skal kjøres i det grafiske brukergrensesnittet til VS Code. Testdrevet utvikling (TDD) vil si at vi skriver tester først, deretter koden, og til slutt forbedrer (refaktorerer) den. Med parametere kan en enkelt test kjøres flere ganger med forskjellige data. Senere skal vi også lære om hvordan vi tester unntak (*exceptions*), bruker *fixtures* for å unngå duplisering av kode og *mocking* for å simulere objekter.

### Øvingsoppgaver

#### 2.5.1

Bruk testdrevet utvikling til å lage funksjonen `bmi_klassifisering(høyde:float, vekt:float) -> str`. Denne funksjonen skal returnere klassifiseringen basert på høyde og vekt. BMI beregnes som vekt/(høyde\*høyde), hvor høyden er oppgitt i **meter** og vekten i **kg**. Argumentene skal være desimaltall med to desimaler for høyden og én desimal for vekten. Vi antar at argumentene allerede er kontrollert for å være realistiske og av riktig type før de brukes i funksjonen, så vi trenger ikke å teste dette i oppgaven. Vennligst lever hele mappen med alle filene i en zippet fil.

Klassifisering	BMI
Undervekt	18,4 eller lavere
Normalvekt	18,5-24,9
Overvekt	25,0-29,9
Fedme, grad 1	30-34,9
Fedme, grad 2	35-39,9
Fedme, grad 3	40,0 eller høyere

<sup>19</sup> Vi skal bruke *pytest-mock* som er både enklere å bruke og lettere å lese enn monkeypatching

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

#### 2.5.2

Vi ønsker å lage et program der brukeren skal skrive inn en poengsum fra og med 0 til og med 100, og programmet skal bestemme og skrive ut tilsvarende karakter etter følgende regler:<sup>20</sup>

- Enhver poengsum under 50 får karakteren «Ikke bestått».
- Enhver poengsum fra og med 50 til og med 69 får karakteren «Bestått».
- Enhver poengsum fra og med 70 til og med 89 får karakteren «Godt bestått».
- Enhver poengsum på 90 eller høyere får karakteren «Meget godt bestått».
- Enhver poengsum som er mindre enn 0 eller større enn 100, får karakteren «Ikke gyldig poengsum!».

Bruk den foreslalte funksjonen nedenfor, gjennomfør testene som er beskrevet i den følgende testplanen, og fyll inn de faktiske resultatene du får. Ta tiden du bruker for hver deloppgave.

- a) Ved hjelp av `print`-setninger
- b) Ved hjelp av `assert`-setninger
- c) Ved hjelp av **pytest** med parametere

```
def karakter(poengsum:int)->str:  
    if poengsum < 50:  
        return('Ikke bestått')  
    elif 50 < poengsum < 69:  
        return('Bestått')  
    elif 70< poengsum < 89:  
        return('Godt bestått')  
    elif 90 < poengsum < 100:  
        return('Meget godt bestått')  
    else:  
        return('Ikke gyldig poengsum!')
```

Test nr.	Input-verdier	Forventet resultat	Faktisk resultat
1	Poengsum: 30	Ikke bestått	
2	Poengsum: 65	Bestått	
3	Poengsum: 82	Godt bestått	
4	Poengsum: 97	Meget godt bestått	
5	Poengsum: 102	Ikke gyldig poengsum!	
6	Poengsum: 0	Ikke bestått	
7	Poengsum: 50	Bestått	
8	Poengsum: 69	Bestått	
9	Poengsum: 70	Godt bestått	
10	Poengsum: 89	Godt bestått	
11	Poengsum: 90	Meget godt bestått	
12	Poengsum: 100	Meget godt bestått	
13	Poengsum: -1	Ikke gyldig poengsum!	

Se filene `karakter_modul.py` og `testresultater.xlsx`.

Vennligst lever hele mappen med alle filene i en zippet fil.

<sup>20</sup> Tilsvarende deler av Oppgave 6 på IT-2 eksamen våren 2023

## 2.6 Objektorientert programmering

### Bokens innhold

Innledning .....	111
Konstruktøren .....	112
Objeksamlinger og ordbøker .....	113
Arv .....	113
Polyformi.....	114
Tilgang .....	114
Innkapsling.....	115
Abstraksjon .....	116
Klassevariabler .....	117
Klassemetoder. ....	117
Modellering .....	118
Ta med minst dette.....	118
Øvingsoppgaver.....	119

### Kompetanse mål:

- anvende objektorientert modellering til å beskrive et programs struktur
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode
- utforske og vurdere alternative løsninger for design og implementering av et program

### Innledning

Ole-Johan Dahl og Kristen Nygaard utviklet programmeringsspråket Simula for Norsk Regnesentral på 1960-tallet. Simula regnes som det første objektorienterte programmeringsspråket. For dette arbeidet ble Dahl og Nygaard tildelt både von Neumann-medaljen og Turing-prisen, som betraktes som informatikkens nobelpris.

I objektorientert programmering tenker vi på verden som bestående av objekter, som kan være alt fra personer, paraplyer, kassaapparater og kortstokker til banklån, bildefiler, farger og følelser. Programmet vi lager prøver å modellere de objektene vi trenger for å løse oppgaven. Som vi har sett, koder vi med klasser, objekter, metoder og attributter. Det er i analyse- og designfasen vi finner ut hvilke klasser, objekter, metoder og attributter vi trenger.

En **klasse** er en oppskrift for hvordan objektene skal være. **Objektene** kan inneholde metoder og attributter. **Metoder** i Python er funksjoner inni en klasse som har med den obligatoriske parameteren **self**. **self** er objektets referanse til seg selv. Vi trenger ikke å ha med

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

`self` som argument når vi kaller metoden. **Attributter** er variabler inni et objekt som tilordnes verdier. Vi kan ha objekter fra forskjellige klasser og vi kan ha flere objekter fra samme klasse. Forskjellige data i attributtene skiller objekter av samme klasse fra hverandre. Hvis vi har to variabler som peker på hvert sitt objekt med likt innhold i attributtene, refererer variablene fortsatt til to forskjellige objekter. Disse instansene ligger lagret på to forskjellige steder. Så når vi skal sammenlikne et objekt med et annet, må vi vite om vi sammenlikner referansene eller innholdet i attributtene.

En klasse kan opprettes med tre ord. Her lager vi klassen Person (`class Person:` `Person`), som gjør ingen ting. Så oppretter vi et objekt av klassen (`Person()`), som vi tilordner variablen `person`. Deretter oppretter vi et attributt i forbifarten som vi kaller `navn`, som vi skriver ut i siste kodelinje. Vi ønsker ikke at attributter skal kunne opprettes fra hvor som helst og kan forhindre dette ved bruk av `__slots__`. Inntil videre lar vi denne problemstilling ligge.

```
class Person:  
    pass  
  
person = Person()  
person.navn = 'Per'  
print(person.navn) # Per
```

### Konstruktøren

Konstruktøren er en metode som kjøres automatisk når vi oppretter et objekt av en klasse. Vi bruker den til å initialisere objektene, gi attributtene startverdier og eventuelt utføre andre oppgaver. I Python heter konstruktøren `__init__`. I andre språk har konstruktøren vanligvis sammen navnet som klassen.

```
class Person:  
    def __init__(self, navn:str, bosted:str)->None:  
        self.navn = navn  
        self.bosted = bosted  
  
    def hils(self):  
        print(f'Hei, jeg er {self.navn} og kommer fra {self.bosted}.')  
  
personer = []  
personer.append(Person('Elise', 'Bergen'))  
personer.append(Person('Frida', 'Stavanger'))  
personer.append(Person('Gustav', 'Oslo'))  
  
for person in personer:  
    person.hils()  
    print(person.navn, person.bosted)
```

Hei, jeg er Elise og kommer fra Bergen.  
Elise Bergen  
Hei, jeg er Frida og kommer fra Stavanger.  
Frida Stavanger  
Hei, jeg er Gustav og kommer fra Oslo.  
Gustav Oslo

Vi bruker konstruktøren til å opprette alle attributtene (objektvariablene) på ett sted. Dessuten har vi lagd en metode `hils()`, som skriver ut en setning med verdien til attributtene.

Samtidig som vi oppretter objektene med `Person()`, legger vi de inn i en liste med `personer.append()`. Til slutt går vi gjennom lista og kaller metoden `hils()` i objektene. Vi ser at attributtene også er tilgjengelig via punktnotasjon.

Ved å pakke data (attributter) og funksjonalitet (metoder) sammen i klasser/objekter oppnår vi modularitet. Andre sentrale prinsipper i objektorientert programmering er arv, polyformi, innkapsling og abstraksjon.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Objektsamlinger og ordbøker

Når vi håndterer mange objekter av samme klasse i et program, kan det være mer effektivt å lagre objektene i en ordbok i stedet for en liste. Ved å bruke en fornuftig nøkkel i ordboken, kan vi raskt finne og få tilgang til objektene. En ordbok gir oss også muligheten til å unngå duplike nøkler, noe som kan være nyttig for å opprettholde unike identifikatorer som brukernavn eller kontonummer. Koden blir dessuten mer lesbar, lettere å vedlikeholde og enklere å feilhåndtere.

```
class Bruker:  
    def __init__(self, fornavn, etternavn):  
        self.fornavn = fornavn  
        self.etternavn = etternavn  
  
    def fullt_navn(self):  
        return f'{self.fornavn} {self.etternavn}'  
  
# Opprett en ordbok med brukere  
brukere = {}  
brukere['henhan'] = Bruker('Henrik', 'Hansen')  
brukere['nornil'] = Bruker('Nora', 'Nilsen')  
  
# Finn og vis fulnlstendig navn til en bruker  
brukernavn = 'nornil'  
if brukernavn in brukere:  
    bruker = brukere[brukernavn]  
    print(f'Fullt navn for "{brukernavn}": {bruker.fullt_navn()}')  
else:  
    print(f'Bruker med brukernavn "{brukernavn}" finnes ikke.')  
✓ 0:0s  
Fullt navn for "nornil": Nora Nilsen
```

### Arv

Vi så i blokk [1.8 Modellering](#) at vi kunne generalisere klasser og opprette en superklasse hvorfra subklassene arver felles metoder og attributter. Dette medfører gjenbruk av programkode. Her har vi generalisert klassene **Hund** og **Katt** til superklassen **Dyr** som har en felles metode **hils()**. Vi sier at subklassene **Hund** og **Katt** arver metoden **hils()** fra superklassen **Dyr**. Superklassen **Dyr** oppgir vi inni parenteser etter klassenavnet hvor vi oppretter subklassene **Hund** og **Katt**. Konstruktørene til **Hund** og **Katt** sender argumentet **navn** videre til superklassen **Dyr** sin konstruktør med **super().\_\_init\_\_(navn)**.

```
class Dyr:  
    def __init__(self, navn):  
        self.navn = navn  
  
    def hils(self):  
        print(f'Hei, jeg heter {self.navn} og er en ' +  
              f'{self.__class__.__name__.lower()}.')  
  
class Hund(Dyr):  
    def __init__(self, navn):  
        super().__init__(navn)  
  
class Katt(Dyr):  
    def __init__(self, navn):  
        super().__init__(navn)  
  
liste_med_dyr = [Hund('Hans')]  
liste_med_dyr.append(Katt('Kristin'))  
for dyr in liste_med_dyr:  
    dyr.hils()  
  
Hei, jeg heter Hans og er en hund.  
Hei, jeg heter Kristin og er en katt.
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Polyformi

Polyformi betyr at noe kan opptre i flere former. I OOP vil det si at vi kan kalle opp metoder med samme navn, men ulikt innhold i objekter av forskjellige klasser.

På grunn av [duck typing](#) i Python trenger vi ikke å definere metoden `hils()` i superklassen eller et grensesnitt. Vi har likevel tatt den med i superklassen for å gjenbruke noe av koden. Så lenge metoden eksisterer for objektet går det bra.

Hvis metoden eksisterer både i superklassen og subklassen, brukes metoden i subklassen. Den overstyrer (*overrides*) metoden i superklassen. Hvis metoden ikke finnes i subklassen, men bare i superklassen, brukes sistnevnte.

Med arv kan vi gjenbruke eksisterende kode. Med polyformi kan vi bestemme hvilken form koden skal ha for objektet det angår.

Legg merke til at vi ikke trenger å endre programmet når vi introduserer et nytt dyr, `Sau`. Nå fungerer koden som den er:

```
Voff! Jeg heter Hans og er en hund  
Mjau! Jeg heter Kristin og er en katt  
Bæ! Jeg heter Sigurd og er en sau
```

```
class Dyr:  
    def __init__(self, navn:str)->None:  
        self.navn = navn  
  
    def hils(self)->None:  
        print('Jeg heter ' + self.navn +  
              ' og er en ' +  
              self.__class__.__name__.lower())  
  
class Hund(Dyr):  
    def __init__(self, navn:str)->None:  
        super().__init__(navn)  
  
    def hils(self)->None:  
        print('Voff!', end='')  
        super().hils()  
  
class Katt(Dyr):  
    def __init__(self, navn:str)->None:  
        super().__init__(navn)  
  
    def hils(self)->None:  
        print('Mjau!', end='')  
        super().hils()  
  
class Sau(Dyr):  
    def __init__(self, navn:str)->None:  
        super().__init__(navn)  
  
    def hils(self)->None:  
        print('Bæ!', end='')  
        super().hils()  
  
liste_med_dyr = []  
liste_med_dyr.append(Hund('Hans'))  
liste_med_dyr.append(Katt('Kristin'))  
liste_med_dyr.append(Sau('Sigurd'))  
for dyr in liste_med_dyr:  
    dyr.hils()
```

Dersom subklassen ikke oppretter en metode for `hils()`, brukes metoden `hils()` i superklassen.

```
Jeg heter Mikke og er en mus
```

Uten polyformi hadde vi vært nødt til å endre koden alle steder hvor den var avhengig av hvilket dyr det var. Nå gjør vi det bare på ett sted når vi oppretter metoden `hils()` i subklassen `Sau`.

```
class Mus(Dyr):  
    def __init__(self, navn:str)->None:  
        super().__init__(navn)  
  
dyr = Mus('Mikke')  
dyr.hils()
```

```
Voff!  
Mjau!  
Bæ!
```

```
for dyr in liste_med_dyr:  
    if isinstance(dyr, Hund):  
        print('Voff!')  
    elif isinstance(dyr, Katt):  
        print('Mjau!')  
    elif isinstance(dyr, Sau):  
        print('Bæ!')
```

### Tilgang

I andre programmeringsspråk, som Java og C#, kan vi styre tilgangen til en klassens variabler og metoder med "synlighetsmodifikatorer" (*access modifiers*). Dette gjøres med de reserverte ordene *public*, *private* og *protected*, som skrives foran deklarasjonene.

- **(standard)** : Synlig i pakken (, klassen og subklassen)
- **public** : Synlig for alle
- **private** : Synlig kun i klassen
- **protected** : Synlig i pakken og i subklasser

Disse ordene finnes ikke i Python hvor vi ikke deklarerer variabler og i utgangspunktet kan sette et attributt i et Python-objekt til hva vi vil, hvorfra vi vil.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Innkapsling

Innkapsling hjelper oss med å kontrollere tilgangen til en klassens attributter og metoder ved å skjule klassens interne detaljer. I mange programmeringsspråk oppnår vi dette ved å markere attributter og metoder som `private` eller offentlige (`public`). Ved å gjøre attributtene private og kombinere dem med offentlige `setter`- og `getter`-metoder, sikrer vi at verdien til et attributt ikke kan endres eller nås direkte utenfra. For eksempel kan vi bruke `hent_variabelnavn()` for å hente verdien til et attributt, og `sett_variabelnavn(verdi)` for å endre den. Dette gir oss muligheten til å kontrollere at dataene er korrekte før de lagres, for eksempel for å sikre at en alder er et positivt tall. Uten offentlige `getter`- og `setter`-metoder forblir private attributter skjulte og utilgjengelige utenfra. Python støtter ikke `private` og `public`, men vi kan oppnå lignende funksjonalitet ved bruk av enkel eller dobbel understrek foran variabelnavnet, eller ved å bruke `@property`-dekoratøren.

1. Ved å bruke en enkel understrek, som i `_alder`, signaliserer vi til andre programmerere at variabelen er ment for internt bruk, selv om den teknisk sett er tilgjengelig utenfra.

```
class Person:  
    def __init__(self, alder:int)->None:  
        self._alder = alder # public  
  
    person = Person(20)  
    print(person._alder)  
    person._alder = -14  
    print(person._alder)  
    ✓ 0.0s  
    20  
    -14
```

2. En dobbel understrek, som i `__alder`, fører til at Python endrer navnet på variabelen for å gjøre den vanskeligere å nå utenfra. Dette kan kombineres med metoder som `hent_alder()` og `sett_alder()` for å validere verdiene før de tilordnes variabelen. Den kan fortsatt nås direkte, men det krever en spesiell syntaks: `_Klassenavn__variabelnavn`, for eksempel `_Person__alder`.

```
class Person:  
    def __init__(self, alder:int)->None:  
        self.__alder = alder # for internt bruk  
  
    person = Person(20)  
    print(person.__alder) # har likevel lesetilgang  
    person.__alder = -14 # bryter anbefalt praksis  
    print(person.__alder) # ingen kontroll på verdi  
    ✓ 0.0s  
    20  
    -14
```

```
class Person:  
    def __init__(self, alder: int) -> None:  
        self.__alder = alder # navnendring, alder er privat  
  
    def hent_alder(self) -> int: # getter  
        return self.__alder  
  
    def sett_alder(self, alder: int) -> None: # setter  
        self.__alder = alder # her kan vi legge inn kontroll  
  
    person = Person(alder=18) # privat alder  
    print(person.hent_alder()) # getter  
    person.sett_alder(19) # setter kan ha kontroll  
    print(person.hent_alder()) # getter  
    person.__Person__alder = 20 # kan likevel omgå setter.  
    print(person.hent_alder()) # getter  
    try:  
        print(person.__alder) # normal notasjon gir feil  
    except Exception as e:  
        print(e.__class__.__name__, e)
```

```
18  
19  
20  
AttributeError 'Person' object has no attribute '__alder'
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

3. Med `@property` og `@variabelnavn.setter` lager vi en *getter*- og en *setter*-metode for variablene. Dette lar oss bruke punktnotasjon utenfra, som `person.alder`, men det er faktisk metodene som blir kalt. Selv med denne tilnærmingen kan den interne representasjonen fortsatt nås, for eksempel ved å finne det hemmellijige variabelnavnet via `person.__dict__.keys()`.

I Python tilbyr `@property`-dekoratøren og den tilhørende `@variabelnavn.setter` en elegant og effektiv måte å oppnå innkapsling på sammenlignet med tradisjonelle *getter*- og *setter*-metoder. Samtidig kan variablene skjules ved å gjøre dem *private* med to understreker (`_`) foran variabelnavnet. Husk samtidig at alt kan sees i Python hvor det er en kultur for åpenhet, så hvis du er i tvil, bruk `public`.

### Abstraksjon

Abstraksjon i OOP vil si å skjule unødvendige detaljer for å gjøre det enklest mulig for brukere av programmet. Tenk på programmet som en svart boks. Brukerne er mer opptatt av hva boksen gjør enn hvordan den det gjør det. De metodene og attributtene vi gjør tilgjengelige for omverden kaller vi for programmeringsgrensesnittet (*API - Application Programming Interface*). Dette kan andre brukere (utviklere) bruke i sine programmer for å gjenbruke vår kode. Hva vi velger å vise og hva vi velger å skjule, styrer vi med overnevnte synlighetsmodifikatorer. Hvis vi sammenligner et program med en TV, vil fjernkontrollen være programmets API. Den som bruker TV-en, velger kanaler og justerer lyden med fjernkontrollen uten å bry seg om hvordan TV-ens elektronikk er konstruert og virker.

```
class Person:
    def __init__(self, alder: int) -> None:
        self._alder = alder # Settes via alder.setter

    @property # metoden kan kalles uten parenteser (getter)
    def alder(self) -> int:
        return self._hemmelig_alder

    @alder.setter # metoden kalles uten parenteser (setter)
    def alder(self, alder: int) -> None:
        # Validatorer og setter alderen
        try:
            alder = int(alder)
        except Exception as e:
            raise type(e)(f'{e.__class__.__name__}: "{str(alder)}" er ikke et heltall.')
        if alder < 0:
            raise ValueError(f'ValueError: {str(alder)} må være 0 eller positivt.')
        self._hemmelig_alder = alder # Lagrer den validerte verdien internt

person = Person(18) # __init__ kaller self.alder(18)
print(person.alder) # kaller person.alder()
person.alder = 19 # kaller person.alder(19)
print(person.alder) # kaller person.alder()

# Sett alder til noe annet enn et heltall
try:
    person.alder = 'tjue'
except Exception as e:
    print(e) # ValueError blir fanget opp og skrevet ut

# Sett alder til noe annet enn et heltall
try:
    person.alder = [1,2,3]
except Exception as e:
    print(e) # TypeError blir fanget opp og skrevet ut

# Sett alder til et negativt tall
try:
    person.alder = -20
except Exception as e:
    print(e) # Vår egen ValueError blir fanget opp og skrevet ut

# Vis tilgjengelige attributter, inkludert den private __hemmelig_alder
print(person.__dict__.keys())

# Endre den private variablene direkte, noe som bryter med innkapsling
person._Person__hemmelig_alder = 20
print(person.alder)
```

```
18
19
ValueError: "tjue" er ikke et heltall.
TypeError: "[1, 2, 3]" er ikke et heltall.
ValueError: -20 må være 0 eller positivt.
dict_keys(['_Person__hemmelig_alder'])
20
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Klassevariabler

Attributtene vi har omhandlet til nå har vært variabler koblet til objektene, såkalte

**objektvariabler**. Noen ganger trenger vi variabler som er koblet til selve klassen. La oss si vi produserer pistoler og ønsker å gi hver pistol et serienummer som begynner på 1 for den første pistolen. Serienummet er unikt for hver pistol og dermed en objektvariabel. Samtidig må vi holde styr på hvor mange pistoler vi har produsert så vi vet hvilket serienummer vi skal tildele neste pistol vi produserer. Denne informasjonen er ikke koblet til en enkelt pistol, men til klassen som helhet. Når vi oppretter en variabel inni en klasse, men ikke inni en metode, blir det **klassevariabel** (*static*) i Python.

Klassevariabler refereres gjennom en klasse, men ikke direkte gjennom et objekt.

```
print(f'\nAntall pistoler: {Pistol.antall_pistoler}') Antall pistoler: 4
```

Når vi bruker `print`, kaller Python opp `str`, som igjen kaller `__str__`. Det som skrives ut er oftest til liten nytte for objekter, se over. Vi kan være mer interessert i å se verdiene til objektvariablene og kan få til dette ved å lage vår egen `__str__`-metode i `Pistol`-klassen.

```
def __str__(self):
    return 'Denne pistolen har serienummer ' \
           + str(self.serienummer) + '.'

for pistol in pistoler:
    print(pistol)
```

```
<__main__.Pistol object at 0x00000243C6B821A0>
```

```
Denne pistolen har serienummer 1.
Denne pistolen har serienummer 2.
Denne pistolen har serienummer 3.
Denne pistolen har serienummer 4.
```

### Klassemetoder.

På samme måtes som det finnes klassevariabler finnes det også klassemetoder, men de brukes ikke så ofte objektmetoder. Klassemetoder har tilgang til klassevariabler, men ikke til objektvariabler. Klassemetoder blir blant annet brukt til å forenkle opprettelsen av objekter. Vi kan eksempelvis ha en metode koblet til klassen som leser data fra en fil og bruker verdiene som argumenter når vi lager nye objekter. Vi lager en klassemetode ved å skriv `@classmethod`-dekoratøren på linjen over funksjonsdefinisjonen. På samme måte som vi har med `self` som den første parameteren i en objektmetode, har vi med `cls`, som den første parameteren i en klassemetode. Når vi kaller opp en klassemetode, skriver vi klassenavnet foran funksjonsnavnet og vi trenger ikke å ha med `cls` i argumentene.

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
@classmethod
def fra_csv(cls, datafil:str)->list:
    liste_med_dyr = []
    with open(datafil,"r",encoding='utf-8') as f:
        linjer = csv.reader(f)
        for linje in linjer:
            navn, type = linje
            if type=='Hund': liste_med_dyr.append(Hund(navn))
            elif type=='Katt': liste_med_dyr.append(Katt(navn))
    return liste_med_dyr
```

```
liste_med_dyr = Dyr.fra_csv('b_2_6_7_data.csv')
for dyr in liste_med_dyr:
    dyr.hils()
```

b_2_6_7_data.csv
1 Hans,Hund
2 Hege,Hund
3 Kristin,Katt
4 Kåre,Katt
5 Knut,Katt

```
Voff! Jeg heter Hans og er en hund
Voff! Jeg heter Hege og er en hund
Mjau! Jeg heter Kristin og er en katt
Mjau! Jeg heter Kåre og er en katt
Mjau! Jeg heter Knut og er en katt
```

## Modellering

```
@startuml Synlighet
skinparam classAttributeIconSize 0
class Klassenavn {
    {static} +klasseVariabel : DataType
    {static} #klasseVariabel : DataType
    {static} -klasseVariabel : DataType

    +objektVariabel : DataType
    #objektVariabel : DataType
    -objektVariabel : DataType
    <<hent>> objektVariabel : DataType {readOnly}
    <<hent, sett>> objektVariabel : DataType
    <<property>> objektVariabel : DataType {readOnly}
    <<property>> objektVariabel : DataType

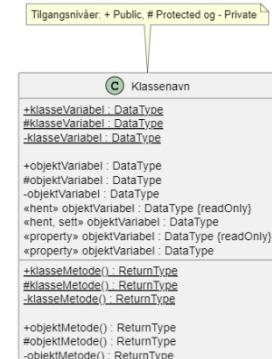
    {static} +klasseMetode() : ReturnType
    {static} #klasseMetode() : ReturnType
    {static} -klasseMetode() : ReturnType

    +objektMetode() : ReturnType
    #objektMetode() : ReturnType
    -objektMetode() : ReturnType
}

note top of Klassenavn
Tilgangsnivåer: + Public, # Protected og - Private
end note
@enduml
```

Når vi lager klassediagram i *plantUML*, skriver vi **{static}** foran klassevariabler og klassemetoder. De vises da med understrek i henhold til spesifikasjonen.

Hvis vi ikke bruker noen synlighetsmodifikator i Python, er variablene *public*, og vi bruker **+** (plussstegnet) i PUML. Hvis vi bruker én understrek foran variabelnavnet i Python tilsvarer det *protected*, og vi setter **#** (nummertegnet) foran i PUML. Hvis vi bruker dobbel understrek foran variabelnavnet i Python, kan det sammenlignes med *private*, og vi setter **-** (minustegnet) foran navnet i PUML.



Vi ønsker ikke å gjøre UML-klassediagrammene våre for kompliserte ved å inkludere tre linjer for hvert attributt som har *getter*- og *setter*-metoder. Selv om det ikke finnes en egen definisjon for dette i UML, kan vi lage vår egen stereotype, **<<get, set>>** eller **<<hent, sett>>**, som vi setter foran variabelnavnet. Dette indikerer tilstedevarelsen av både en *get*- og *set*-metode eller *hent*- og *sett*-metoder. For Python kan vi spesifikt bruke **<<property>>**, som for eksempel **<<property>> alder: int**. Hvis det kun er mulig å lese variablen, altså det bare finnes en *getter* og ingen *setter*, kan vi legge til **{readOnly}** for å vise dette. Eksempelet blir da **<<property>> alder: int {readOnly}**.

## Ta med minst dette

**Objekter** er sammensatte datatyper som inneholder data i form av attributter og kode i metoder. Disse objektene skapes fra **klasser**, definert med **class Klassenavn:**, som fungerer som oppskrifter for å opprette objekter. En klasse spesifiserer hvilke **attributter** (variabler) og **metoder** (funksjoner) objektene skal inneholde. Vi skiller mellom attributter som tilhører individuelle objekter og klassevariabler som er felles for alle objektene i klassen.

**Konstruktøren** **`__init__(self,...)`** er en spesiell metode som kjøres hver gang et objekt opprettes.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

`self` refererer til objektet som blir opprettet og må brukes foran alle attributtene for å gjøre dem tilgjengelige for objektets metoder. Metoder som kalles fra andre metoder innenfor samme objekt, må også ha `self` foran metodenavnet. Vi inkluderer ikke `self` som argument når vi oppretter et objekt med klassenavnet etterfulgt av parenteser og eventuelle argumenter. Vi oppnår en rekke fordeler dersom vi organiserer objektsamlinger i **ordbøker** fremfor lister. Når vi implementerer **arv**, angis superklassen inni parentesene i klassedefinisjonen, noe som gjør at en subklasse kan arve attributtene og metodene fra superklassen. Dette muliggjør gjenbruk av kode og støtter **polyformi**, som er evnen til å behandle objekter fra forskjellige klasser på samme måte. **Abstraksjon** forenkler bruken av klasser ved å fokusere på hvilke attributter og metoder som skal eksponeres for omverdenen. **Innkapsling** beskytter attributter og metoder ved å skjule dem, og **synlighetsmodifikatorer** styrer tilgangen til klassens attributter og metoder. Litt forenklet kan vi si at vi modellerer abstraksjon og programmerer det som innkapsling ved hjelp av synlighetsmodifikatorer.

#### Øvingsoppgaver

##### 2.6.1

Lag et program som kan opprette bankkontoer. Bankkontoene skal ha eier, kontonummer og saldo, som er beløpet som til enhver tid står på kontoen. Når bankkontoen opprettes, skal saldoen settes til 0. Det skal gå an å sette inn og ta ut penger fra kontoen. Når penger settes inn/tas ut, skal programmet kvittere med beløpet som ble satt inn/tatt ut og ny saldo. Det skal ikke være mulig å ta ut mer penger enn det som står på kontoen. Et slikt forsøk skal kvitteres med en feilmelding. Du trenger bare levere ett program.

a)

Lag en modell oppgaven med et klassediagram.

b)

Lag programmet. Eksempel på bruk og utskrift er gitt vist under.

```
konto = Konto('Hege Hermansen', '1020.30.45678')
konto.sett_inn(1000)
konto.sett_inn(2000)
konto.ta_ut(1500)
konto.ta_ut(2500)
```

```
Nyopprettet konto:
Eier      : Hege Hermansen
Kontonummer: 1020.30.45678
Saldo    : 0

Du satt inn 1000.
Ny saldo er 1000

Du satt inn 2000.
Ny saldo er 3000

Du tok ut 1500
Ny saldo er 1500

Du prøver å ta ut 2500.
Ikke dekning på konto
Saldoen er 1500
```

c)

I de videre oppgavene, kan du støtte deg til artikkelen [8 Tips for Object-Oriented Programming in Python](#). Hvis du ikke har gjort det alt, opprett attributtene i konstruktøren

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

(Tips #1). Lag din egen skreddersydde versjon av `__str__` metoden som skriver ut en brukervennlig versjon av verdiene i konto-objektet (Tips #3).

```
print(konto)
```

Konto:	Eier : Hege Hermansen
	Kontonummer: 1020.30.45678
	Saldo : 1500

d)

Beskytt attributtet `saldo` så det blir *private*, tilgjengelig via punktnotasjon og kun *readOnly* (Tips #7)

```
print(f'Saldo: {konto.saldo}')  
konto.saldo = 10000
```

Saldo: 1500

AttributeError: can't set attribute 'saldo'

e)

Opprett en privat klassevariabel med navnet `_MIN_SALDO` og verdien -5000, som angir det maksimale beløpet som tillates å overtrekke kontoen. (Tips #2).

```
konto = Konto('Hege Hermansen', '1020.30.45678')  
konto.sett_inn(1000)  
konto.sett_inn(2000)  
konto.ta_ut(5000)  
konto.ta_ut(4000)
```

Konto:	Eier : Hege Hermansen
	Kontonummer: 1020.30.45678
	Saldo : 0
Du satt inn 1000.	
Ny saldo er 1000	
Du satt inn 2000.	
Ny saldo er 3000	
Du tok ut 5000	
Ny saldo er -2000	
Du prøver å ta ut 4000.	
Ikke dekning på konto	
Saldoen er -2000	

f)

Lag en offentlig klassemetode `fra_csv` som leser inn kontodata (eier og kontonummer) fra en fil og oppretter kontoen (Tips #2)

```
konto = Konto.fra_csv('p_2_6_data.csv')
```

> p\_2\_6\_data.csv  
Hege Hermansen,1020.30.45678

Konto:	Eier : Hege Hermansen
	Kontonummer: 1020.30.45678
	Saldo : 0

g)

Oppdater Klassediagrammet til å skille mellom klasse- og objekt attributter og metoder samt synligheten til disse (*public*, *protected*, *private*).

## 2.7 Noen enkle verktøy for systemutvikling

### Bolkens innhold

Innledning .....	121
Pseudokode .....	121
Flytskjema med drawio.....	123
Wireframes med Pencil .....	124
Ta med minst dette.....	126
Øvingsoppgaver.....	126

### Kompetansemål:

- velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

### Innledning

I dette [vedlegget](#) er det en oversikt over verktøy og metoder for systemutvikling som vi har valgt for å oppfylle kompetansemålet, se [KM09](#). Vi har allerede lært om noen av disse verktøyene. I denne bolken gjennomgår vi **pseudokode**, **flytskjema** og **wireframes**. De resterende verktøyene vil bli dekket i den siste runden. I 4.7 Flere verktøy for samarbeid, skal vi lære om *Live Share*, *Slack*, *Trello* og *GitHub* og i 4.9 Fortsettelse av objektorientert modellering, skal vi lære om brukstilfelle- og sekvensdiagram, se også [KM04](#). Det finnes naturligvis andre verktøy med tilsvarende funksjoner.

Pseudokode, flytskjema og wireframes er verktøy som brukes innen systemutvikling for å beskrive algoritmer eller planer for et system på en enkel og forståelig måte. Pseudokode er en mellomting mellom algoritmer og faktiske dataprogrammer. Det har røtter tilbake til 1950-tallet og tidligere former for lavnivåspråk. Det ble utviklet for å gi en generell beskrivelse av en algoritme, med et mer naturlig språk enn programmeringsspråk. Flytskjema ble oppfunnet på 1920-tallet som et verktøy for å visualisere arbeidsflyt i industriproduksjon, men ble raskt tatt i bruk som et verktøy for å beskrive algoritmer på en visuell og forståelig måte. Wireframes kan spores tilbake til grafisk design så tidlig som på 1920-tallet, ble tatt i bruk innenfor prototyping på 1980-tallet og webdesign på 1990-tallet. Wireframes er skisser som illustrerer brukergrensesnittet ved å tydelig visualisere plasseringen av elementer og funksjonalitet på en oversiktlig måte.

### Pseudokode

Husk at en algoritme er et sett med instruksjoner eller en oppskrift som beskriver hvordan vi løser et bestemt problem. En algoritme kan utføres av en datamaskin, men den kan også utføres manuelt. Et dataprogram er en samling av instruksjoner som datamaskinen kan forstå og utføre. Det kan inneholde én eller flere algoritmer. Så mens algoritmer er en abstrakte

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

beskrivelser av hvordan vi skal løse et problem, er et dataprogram en konkret realisering av disse algoritmene.

Det finnes mange forskjellige typer programmeringsspråk med detaljerte egenskaper som må følges til punkt og prikke når vi koder algoritmen. Dette gjør det vanskelig å lese og forstå koden for mennesker. Det er her pseudokode kommer inn i bildet. Pseudokode er en generell beskrivelse av en algoritme, skrevet med et mer naturlig språk enn programmeringsspråk. Pseudokode er en mellomting mellom en algoritme og et faktisk dataprogram, og er et nyttig verktøy for å forbedre kommunikasjon, samarbeid og forståelse mellom utviklere, samt for å lære nybegynnere programmering.

Selv om pseudokode ikke er et formelt, standardisert språk, finnes det en standard for sentralt gitt IT-2 eksamen. Den er beskrevet i gjeldende eksamsveiledning (2024), og det forventes at kandidatene behersker denne standarden på eksamen. Eksamensveiledningen utgis årlig, så dette kan endres. I 2024 innebærer dette at standardiserte nøkkelord skrives med STORE BOKSTAVER på engelsk, mens resten av pseudokoden kan være på norsk. Det forventes at studentene bruker denne standarden konsistent, og at de skriver med en klar og presis stil som er lett å forstå. Alle strukturblokker skal være indentert i pseudokode. De grunnleggende programmeringskonseptene vi lærte om i den første bolken, vil kunne se slik ut med denne standarden:

- Inndata : READ alder
- Utdata : DISPLAY "Du er " + alder + "år."
- Beregninger : SET total TO antall \* pris
- Valgsetninger :
  - IF alder GREATER THAN OR EQUAL TO 18
    - DISPLAY "Du er myndig"
    - ELSE
      - DISPLAY "Du er ikke myndig"
    - ENDIF
- Løkker :
  - FOR hver n LESSER THAN OR EQUAL TO 10
    - DISPLAY n
  - ENDFOR

Vi trenger ingen spesielle verktøy for å skrive pseudokode og kan bruke et vanlig tekstbehandlingsprogram, en teksteditor eller en tavle for den saks skyld.

Et eksempel på en algoritme for å finne det største tallet i en liste, kan se slik ut:

1. Sett maks til det første tallet i listen.
2. Gå gjennom hver verdi i listen fra og med den andre tallet.
3. Hvis tallet i listen er større enn maks, sett maks til det tallet.
4. Når alle tallene i listen er sjekket, vil maks inneholde det største tallet.
5. Skriv ut maks.

Hvis vi gjør det om til standard pseudokode, kan se slik ut:

```
READ liste
SET maks TO første tall i liste
FOR hvert tall i liste
    IF tall GREATER THAN maks
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
SET maks TO tall  
ENDIF  
ENDFOR  
DISPLAY maks
```

En av de største fordelene med pseudokode er at det kan være enkelt å oversette til faktisk programkode, uansett hvilket programmeringsspråk som brukes.

I Python og JavaScript kan programmet til pseudokoden realiseres som følger:

Python:

```
liste = [3,7,1,9,4,2,8,6,5]  
maks = liste[0]  
  
for tall in liste:  
    if tall > maks:  
        maks = tall  
  
print(maks)
```

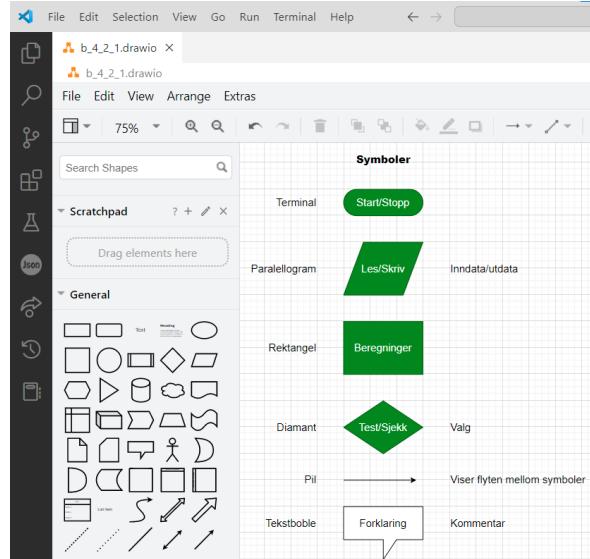
JavaScript:

```
let liste = [3,7,1,9,4,2,8,6,5];  
let maks = liste[0];  
for (let i = 1; i < liste.length; i++) {  
    let tall = liste[i];  
    if (tall > maks) {  
        maks = tall;  
    }  
}  
console.log(maks);
```

Alt i alt er pseudokode en effektiv måte å beskrive en algoritme på, fordi det er enklere og mer naturlig å lese enn formelle programmeringsspråk. Dette gjør pseudokode til et nyttig og fleksibelt verktøy for å utveksle ideer og samarbeide i systemutvikling. Dessuten er det nyttig for nybegynnere som ønsker å lære programmering. På nettsiden [Pseudocode Standard](#) finnes flere eksempler på hvordan vi kan bruke en standard pseudokode på en god måte.

### Flytskjema med drawio

Flytskjemaer, også kalt flytdiagrammer, er en grafisk representasjon av en sekvens av handlinger som skal utføres. De er en viktig del av systemutvikling og programmering, og brukes til å beskrive kontrollflyten i et program. Å lage et flytskjema kan hjelpe oss med å planlegge et program på en strukturert måte, og kan være et nyttig alternativ til pseudokode hvis vi foretrekker en mer visuell tilnærming. Ved hjelp av en rekke standardiserte symboler kan vi definere hva slags inndata programmet skal motta, hvilke beregninger som skal utføres, hvilke valg som programmet skal ta og hvilke utdata programmet skal gi.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Det finnes mange verktøy for å lage flytskjema, som for eksempel [Lucidchart](#), [Microsoft Visio](#) og [draw.io](#). Vi skal bruke VS Code-utvidelsen til draw.io. Når vi oppretter en fil med filtypen [.drawio](#), er det enkelt å dra figurer ut på lerret og koble dem sammen med piler, se også [veiledningene](#) til draw.io.

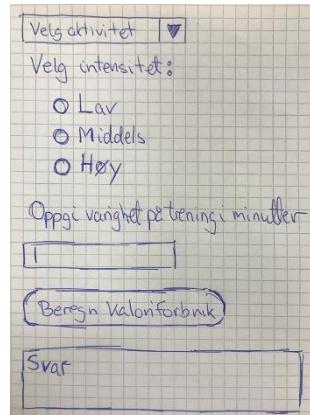
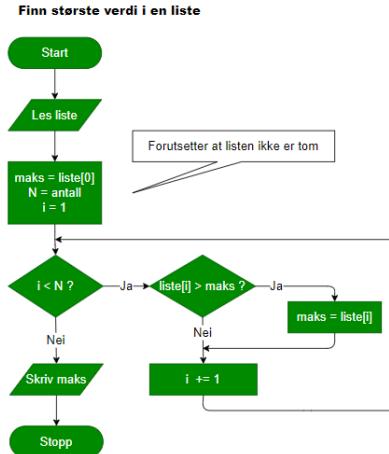
De vanligste symbolene i et flytskjema for et program er terminaler, som er ut som idrettsbaner, for start og stopp, parallelogram for inn- og utdata, rektangler for beregninger og diamanter for valgsettninger. I tillegg må vi ha med piler for å vise kontrollflyten og eventuelle kommentarer. Løkker må vi lage selv med en kombinasjon av symbolene. I figuren til høyre viser vi et flytskjema for å finne den største verdien i en liste. Det viser også hvordan vi kan lage en for-løkke.

Sammenlignet med pseudokode kan flytskjemaer være lettere å forstå og lage, spesielt for de som ikke er kjent med programmering og har mindre forståelse for syntaks og semantikk. Noen foretrekker grafikk fremfor tekst og liker dermed flytskjemaer bedre enn pseudokode, mens det kan være omvendt for andre. Pseudokode kan være et bedre alternativ når man trenger å beskrive detaljerte og komplekse algoritmer som kan være vanskelige å visualisere i et flytskjema.

### Wireframes med Pencil

*Wireframes* er skisser som hjelper oss med å planlegge og kommunisere hvordan en applikasjon skal se ut og fungere for brukeren. I motsetning til pseudokode og flytskjema, som beskriver logikken og algoritmene i et program, representerer *wireframes* utseendet og funksjonaliteten til et brukergrensesnitt. Det gjør det mulig for utviklere, designere, prosjektledere og brukere å samarbeide mer effektivt og oppdage problemer eller uklarheter tidlig i utviklingsprosessen. Det har åpenbare fordeler, for hvis vi må gjøre endringer senere, når mye kode allerede er på plass, blir det så godt som alltid mer arbeid som tar lengre tid og koster mer.

Selv om *wireframes* kan virke enkle i utgangspunktet, er det mange beslutninger som må tas både før vi begynner og når vi skal bestemme oss for detaljer. For eksempel må vi tenke på hvilke enheter produktet skal støtte, og hvordan vi kan få bekreftet hvilke enheter som forventes å bli støttet. Vi må også vurdere hvilke funksjoner som skal tilpasses til hvilke enheter, og hvilke funksjoner som skal fjernes fra enheter der de ikke passer. I tillegg må vi ta hensyn til hvordan designforskjeller på ulike enheter skal håndteres, hvordan skjermorienteringen skal håndteres, hvilke tilbakemeldinger som skal gis for hver handling brukerne utfører, hvilken støtte som skal gis når de fyller ut et skjema, hvordan dynamisk innhold skal håndteres, hvordan størrelsen på og plasseringen av titler og etiketter skal



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

tilpasses plassen vi har til rådighet, om vi skal bruke radioknapper eller *checkboxer*, inkludere verktøytips, og mye mer.

Når vi lager *wireframes* og skal ta stilling til alle disse faktorene, er det viktig å ha erfaring og kunnskap om UX/UI-design (*User Experience/User Interface* eller brukeropplevelse/brukergrensesnitt). Derfor finnes det også spesialister og egne stillinger for UX/UI-designere.

The image shows a job listing from FINN.no. At the top left is the FINN logo with the text "Mullighetenes marked". The main title is "eika.". Below the title is a photograph of a green acorn on a leaf. Underneath the photo are social media sharing buttons for heart, mail, Facebook, and Twitter. The job title is "Vil du bidra til å gjøre forsikring enkelt og nært?". Below the title are several details: "Arbeidsgiver: Eika Forsikring AS", "Stillingsittel: UX/UI designer", "Frist: 20.03.2023", and "Ansattesform: Fast".

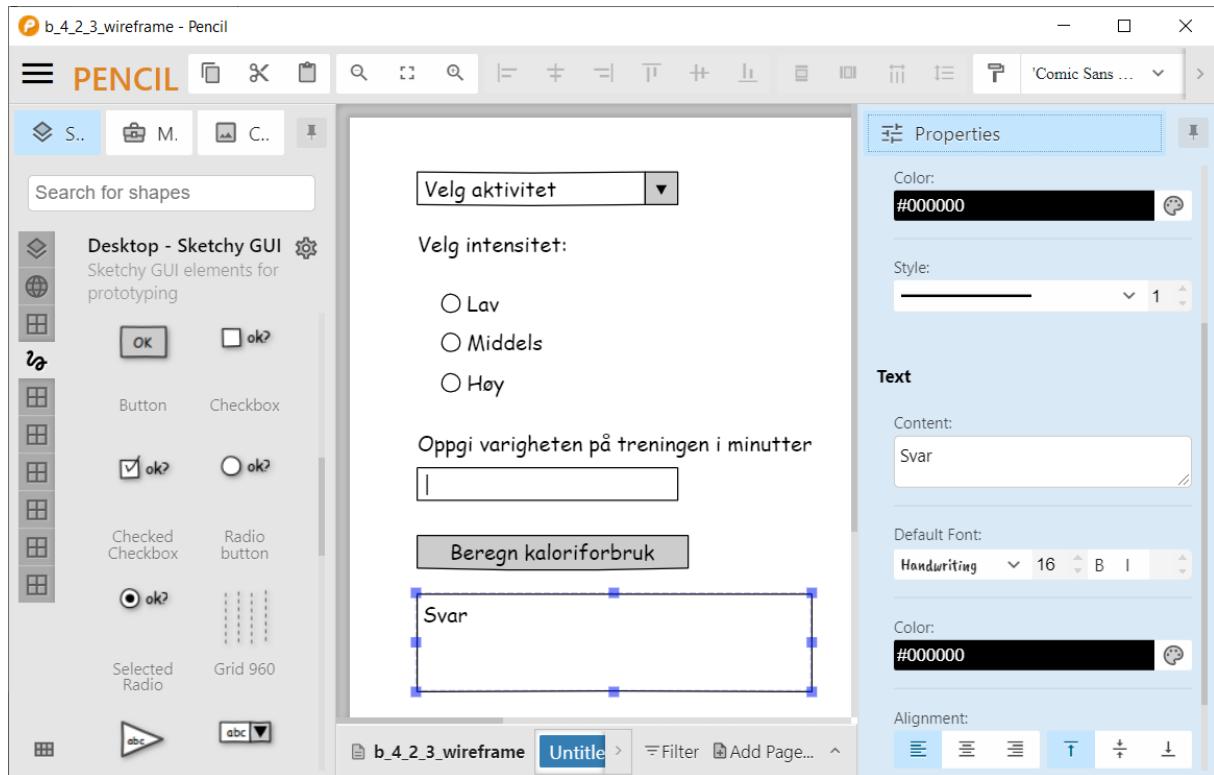
Vi har flere alternativer når vi skal lage wireframes. Vi kan bruke papir og blyant eller en tavle og tegne wireframes for hånd. Det kan være en rask og enkel måte for å eksperimentere med idéer. Som et alternativ kan vi bruke digitale verktøy for å lage wireframes. Disse verktøyene gir vanligvis mer funksjonalitet og mulighet for å lage mer detaljerte design sammenlignet med håndtegninger. Noen populære alternativer er [Sketch](#), [Adobe XD](#), [Figma](#), og [Balsamiq](#). Når vi bruker digitale verktøy, kan vi eksportere wireframes til ulike formater, for eksempel PDF eller bildefiler. Dette gjør det enklere å dele med andre når vi vil diskutere designet. [PlantUML](#), som vi kjenner fra før, kan også brukes til å lage *wireframes*. Se [Salt](#), som er et delprosjekt som inngår i PlantUML.

The image shows a screenshot of a PlantUML code editor. On the left, there is a code editor window containing UML class diagram code. On the right, there is a user interface mockup with various input fields and buttons. The mockup includes a dropdown menu labeled "Velg aktivitet", radio buttons for "Velg intensitet" (Lav, Middels, Høy), a text input field for "Oppgi varighet på trening i minutter", a button labeled "Beregner kaloriforbruk", and a text area labeled "Svar".

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Vi tror imidlertid det er bedre å bruke [Pencil](#), som har et enkelt grafisk brukergrensesnitt og tilstrekkelige symboler til våre formål, og som er gratis.



### Ta med minst dette

**Pseudokode** er en kombinasjon av naturlig språk og elementer fra programmeringsspråk som gjør det enklere å forstå, planlegge og kommunisere algoritmer før de programmeres. Det kan ikke kjøres på en datamaskin, er uavhengig spesifikke programmeringsspråk og kan skrives i hvilket som helst **tekstbehandlingsverktøy**. I 2024 kreves det av kandidatene til sentralt gitt IT-2 eksamen at de behersker en standard for pseudokode som er beskrevet i eksamsveiledningen.

Et **flytskjema** er en grafisk fremstilling av en sekvens med steg som skal utføres i et program. Med standardiserte symboler for start/stopp, inn- og utdata, beregninger, og valg, blir det lettere å forstå og kommunisere kompleks programflyt. Vi har valgt å bruke [draw.io](#)-utvidelsen i VS Code til dette.

**Wireframes** er skisser som hjelper oss med å planlegge og kommunisere hvordan en applikasjon skal se ut og fungere. Vi har valgt å bruke **Pencil** som er et gratis verktøy til å lage tegninger som viser brukergrensesnittets utseende og funksjonalitet.

### Øvingsoppgaver

#### 2.7.1

- Forklar algoritmen i denne pseudokoden.

Sett tilfeldig\_tall til et tilfeldig tall mellom 1 og 100

Sett antall\_forsøk til 0

Mens sann:

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Les tall

Øk antall\_forsøk med 1

Hvis tall er lik tilfeldig\_tall:

Skriv "Gratulerer, du gjettet riktig på", antall\_forsøk, "forsøk"

Bryt løkken

Hvis tall er lavere enn tilfeldig\_tall:

Skriv ut "Tallet er høyere"

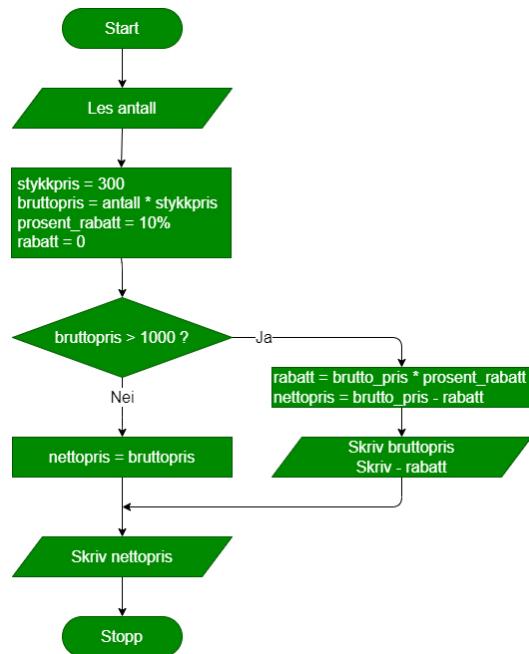
Hvis tall er høyere enn tilfeldig\_tall:

Skriv ut "Tallet er lavere"

- b) Lag et flytskjema for pseudokoden.
- c) Realiser algoritmen i et dataprogram.

#### 2.7.2

- a) Forklar algoritmen i flytskjemaet i figuren til høyre.
- b) Skriv pseudokode for flytskjemaet.
- c) Lag en *wireframe* for programmet.
- d) Realiser algoritmen i et dataprogram.



#### 2.7.3

- a) Skriv pseudokode for å sortere en liste.
- b) Lag tilsvarende flytskjema.
- c) Realiser algoritmen i et dataprogram.

Vurderingsseksemplar

## 2.8 Utveksling av data med JSON

### Bokens innhold

Innledning .....	128
EDI (Electronic data interchange) .....	128
WWW (World Wide Web) .....	129
data.europe.eu.....	129
XML (eXtensible Markup Language) .....	130
JSON (JavaScript Object Notation) .....	131
Ta med minst dette.....	132
Øvingsoppgaver.....	133

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett
- gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

### Innledning

Det er mange grunner til at vi ønsker å utveksle data. Her er noen:

- Handelspartnere kan utveksle dokumenter elektronisk (tilbud, ordre, ordrebekrefte, pakksedler, faktura, fortolling, osv.). Det er billigere enn å skrive ut dokumentene, går raskere enn å sende dem med posten, og vi unngår å registrere den samme informasjonen flere ganger.
- En bedrift kan regelmessige samle inn data fra avdelinger eller datterselskaper i inn- og utland for å konsolidere (oppsummere) data i samlet form (salgstall, regnskapstall, produksjonstall, osv.).
- Når vi surfer på internett, laster vi ned data (websider) fra *webservere* til vår nettleser.
- Tilsvarende har vi apper som etterspør informasjon fra databaser i skyen, eksempelvis når vi bruker Vy-appen til å finne togarter for oss.
- Når vi bytter ut datasystemer må vi normalt måtte overføre data fra det gamle til det nye systemet. Felles for alle disse eksemplene er at datasystemet som sender fra seg data, må gjøre dette på et format som mottakersystemet kan forstå.

### EDI (Electronic data interchange)

I over 50 år har vi hatt ulike **EDI** (*Electronic data interchange*) standarder. Elektronisk utveksling av data foregår ofte mellom kjøpers og selgers datasystemer på et dataformat partene enes om. EDI skiller seg fra e-post fordi e-postmeldinger er ustrukturerte og leses av

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

mennesker. EDI-dokumenter, derimot, har et avtalt og strukturert format som behandles automatisk av datasystemer. [SWIFT](#) er nok et eksempel på EDI, som bankene bruker når vi overfører penger til utlandet.

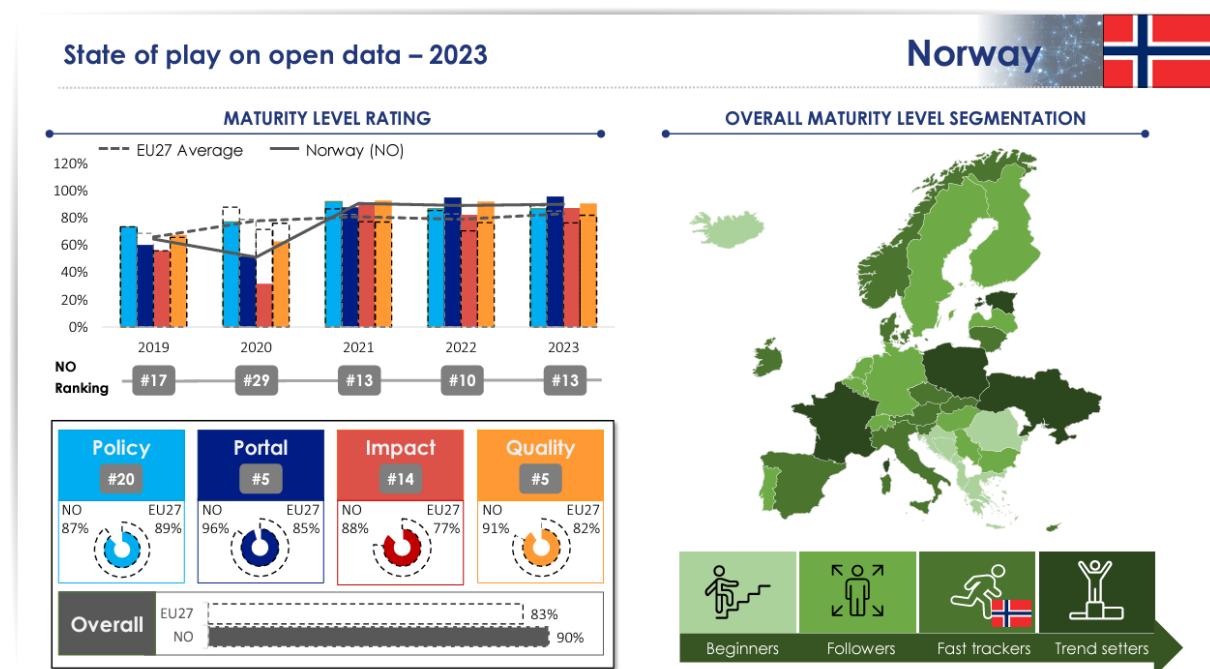
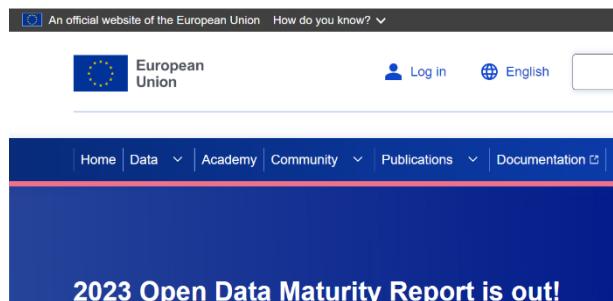
### WWW (World Wide Web)

På begynnelsen av 1990-tallet fikk vi veven (*World Wide Web*) og HTML. Nettsider blir overført fra tjenere (*servers*) til klienter (*clients*) over internett med HTTP-protokollen. En internettprotokoll er regler for hvordan sender og mottaker skal utveksle data. HTML er regler for hvordan disse dataene skal være formatert. Tjenere og klienter kan kjøre forskjellig programvare på ulik maskinvare. Tjeneren kan eksempelvis vær en HP-maskin med Windows og IIS (*Internet Information Server*) eller en Lenovo-maskin med Linux og Apache HTTP Server, samtidig som kliensen kan være en PC med Windows og Chrome eller en Mac med MacOS og Safari.

### [data.europa.eu](#)

Sist vi brukte åpne data, utforsket vi [Felles Datakatalog](#) og [SSB](#). Nå skal vi se hva Europa har å by på: [data.europa.eu](#), tidligere kjent som den det Europeiske Dataportalen, har til formål å fremme tilgang til åpne data på tvers av EU.

Her blir blant annet hvert land rangert etter hvor langt de har kommet med åpne data, se [Norge](#).



Hvis vi søker litt, kan vi finne en oversikt over [UDP - Internet speed at tourism destinations](#), som er en indikator for maksimal tilgjengelig internett-hastighet ved turistdestinasjoner på kommunenivå. Vi har valgt ut data i [JSON-stat](#) format for ulike land. JSON-stat er et format

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

spesialisert for flerdimensjonale statistiske data, som brukes blant annet av nordiske statistikkbyråer, [Eurostats](#) og [data.europe.eu](#). Det kan virke noe komplekst, og selv om det er mulig, er det tungvint å lese dette formatet med standard `json`-biblioteket i Python. Men, biblioteket `pyjstat` kommer oss til unnsetning og gjør det enkelt å få dataene inn i en `pandas DataFrame` for videre analyse og presentasjon i tabeller eller `seaborn`-diagrammer.

Vi leser inn datasettet med `ds = pyjstat.Dataset.read(url)`, og gjør det om til en `DataFrame` med `ds.write('dataframe')` og skriver den ut. Deretter fortsetter vi med `pandas` og `seaborn`. Først kvitter vi oss med fire kolonner. Deretter endrer vi kolonneoverskriftene. Så sorterer vi tabellen og til slutt skriver vi ut et horisontalt stolpediagram med alle destinasjonene.

```
from pyjstat import pyjstat
import seaborn as sns
import matplotlib.pyplot as plt

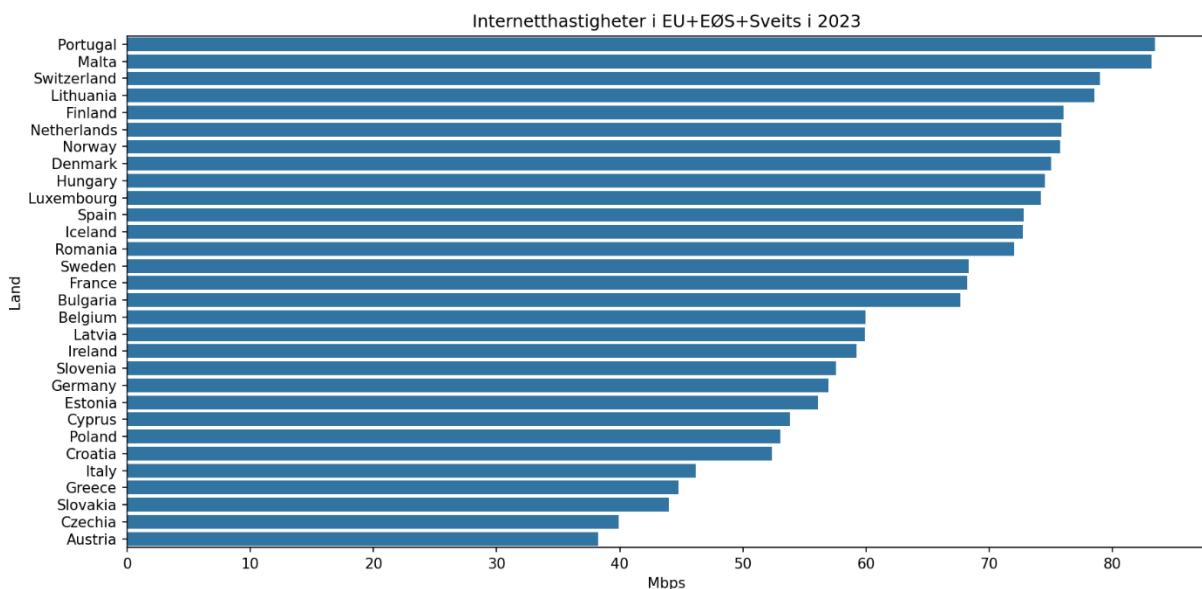
# Les data fra URL-en ved hjelp av pyjstat
url = 'https://urban.jrc.ec.europa.eu/api/udp/v2/en/data/' + \
    '?databrick_id=730&nutslevel=0&year=2023'
dataset = pyjstat.Dataset.read(url)

# Konverter datasettet til en pandas DataFrame
df = dataset.write(output='dataframe')
print(df)
|
# Behold kun kolonnene 'NUTS_CODE' og 'value'
df = df[['NUTS_CODE', 'value']]
df.columns = ['Land', 'Mbps']

# Sorter etter synkende Mbps
df = df.sort_values('Mbps', ascending=False)

# Lag et horisontalt stolpediagram
plt.figure(figsize=(8, 5))
sns.barplot(data=df, x='Mbps', y='Land', orient='h')
plt.title('Internethastigheter i EU+EØS+Sveits i 2023')
plt.tight_layout()
plt.show()
```

	NUTS_CODE	LEVEL_ID	UNIT	VERSION	REF_YEAR	value
0	Austria	0	Mbs	2021	2023	38.22
1	Belgium	0	Mbs	2021	2023	59.95
2	Bulgaria	0	Mbs	2021	2023	67.66
3	Switzerland	0	Mbs	2021	2023	79.00
4	Cyprus	0	Mbs	2021	2023	53.80
5	Czechia	0	Mbs	2021	2023	39.93
14	Hungary	0	Mbs	2021	2023	74.52
15	Ireland	0	Mbs	2021	2023	59.24
16	Iceland	0	Mbs	2021	2023	72.77
17	Italy	0	Mbs	2021	2023	46.19
18	Lithuania	0	Mbs	2021	2023	78.53
19	Luxembourg	0	Mbs	2021	2023	74.22
20	Latvia	0	Mbs	2021	2023	59.92
21	Malta	0	Mbs	2021	2023	83.18
22	Netherlands	0	Mbs	2021	2023	75.85
23	Norway	0	Mbs	2021	2023	75.77
24	Poland	0	Mbs	2021	2023	53.02
25	Portugal	0	Mbs	2021	2023	83.48
26	Romania	0	Mbs	2021	2023	72.02
27	Sweden	0	Mbs	2021	2023	68.34
28	Slovenia	0	Mbs	2021	2023	57.57
29	Slovakia	0	Mbs	2021	2023	43.98



### XML (eXtensible Markup Language)

På slutten av 1990-tallet fikk vi [XML](#). XML er også et markeringsspråk (*markup language*) som HTML, eller rettere sagt en standard for hvordan vi kan lage vårt eget markeringsspråk, hvor vi bestemmer taggene selv. HTML ble laget med tanke på hvordan informasjonen skal vises

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

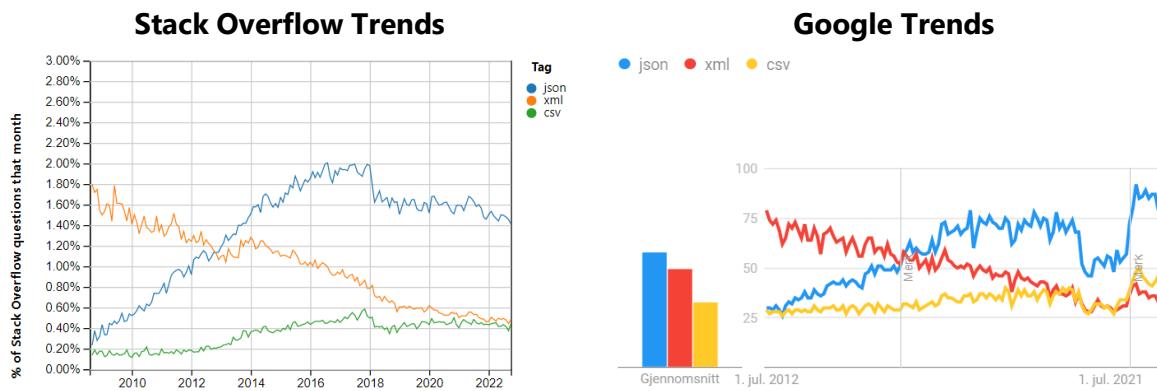
på skjermen. XML er laget med tanke på å beskrive informasjonen. XML gjør ingenting. Det bare beskriver (strukturer og lagrer) data og er derfor godt egnet til datautveksling mellom ulike systemer.

### JSON (JavaScript Object Notation)

På begynnelsen av 2000-tallet fikk vi **JSON** (*JavaScript Object Notation*), som er et enklere format enn XML for datautveksling. Som navnet tilsier, er JSON basert på JavaScript, men det er tekstbasert og støttes av mange programmeringsspråk, deriblant Python.



Siden JavaScript har blitt så populært, har JSON blitt det, også. JSON er enklere å bruke enn XML, men ikke bedre enn XML til alle formål. Klikk på grafene for å gå til referansene.



JSON er basert på to strukturer

- Navn/verdi par, som tilsvarer **dictionary** i Python
- En ordnet liste med verdier som tilsvarer **list** i Python

```
b_2_7_1.json > ...
1 [ ...
2   {"navn": "Knut", "alder": 32},
3   {"navn": "Tiril", "alder": 28},
4   {"navn": "Anne", "alder": 41},
5   {"navn": "Oscar", "alder": 35}
6 ]
```

Her har vi en JSON-fil med en ordnet liste med 4 objekter. [Klammparentesene] omslutter listen, og {krøllparentesene} omslutter objektene. I navn/verdi par må tekstu

være omsluttet av "anførelstegn". 'Apostrofer' godtas ikke. Verdier kan også være objekter, lister, tall, "true", "false" eller "null". Navn/verdi par i objekter adskilles med komma. Det gjør også verdier i lister.

Når vi skal lese inn denne JSON-filen importerer vi først **json**-modulen og åpner filen. Deretter bruker vi **json.load()** som gjør om teksten (i JSON-format) til objekter i Python. Denne prosessen kalles deserialisering eller *parsing* og følger konverteringstabellen som er vist til høyre.

```
import json, pprint
with open('b_2_8_1.json', encoding='utf-8') as fil:
    data = json.load(fil)
print('*** print data(list/dictionaries) ***')
print(data)
print(type(data))
```

[{'navn': 'Knut', 'alder': 32}, {'navn': 'Tiril', 'alder': 28}, {'navn': 'Anne', 'alder': 41}, {'navn': 'Oscar', 'alder': 35}] <class 'list'>

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

Vi ser at vi har fått inn teksten som en liste,

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

```
<class 'dict'>
<class 'dict'>
<class 'dict'>
<class 'dict'>
```

```
for obj in data: print(type(obj))
```

og at listen består av 4 elementer som er ordbøker.

```
print('\nNavnliste:\n*****')
for obj in data:
    for nøkkel, verdi in obj.items():
        print(f'{nøkkel.capitalize():6}: {verdi}')
    print()

Navnliste:
*****
Navn : Knut
Alder : 32

Navn : Tiril
Alder : 28

Navn : Anne
Alder : 41

Navn : Oscar
Alder : 35
```

Til slutt skriver vi ut listen med ordbøkene slik vi lærte i blokken 2.2 Tupler og ordbøker.

Vi må bruke `json.load(fil)`. Hvis vi bruker `fil.read()`, som vi brukte i blokken [1.6 Filer](#), leser vi bare inn en tekst (`str`) og får ingen liste med ordbøker.

Vi kan også gå den andre veien og lagre Python objekter som en tekst i JSON-format i en fil.

```
import json

ordbok = {
    'fornavn': 'Per',
    'yrke': 'Snekker',
    'fødselsår': 1992,
    'poststed': 'Askim'
}

with open('b_2_8_2.json', 'w', encoding='utf-8') as fil:
    json.dump(ordbok, fil, indent=2, ensure_ascii=False)
```

Vi bruker `json.dump()`, og legger til

`indent=2, ensure_ascii=False` for å få en mer lesbar tekstfil. Denne prosessen heter serialisering.

```
{} b_2_8_2.json > ...
1 {
2     "fornavn": "Per",
3     "yrke": "Snekker",
4     "fødselsår": 1992,
5     "poststed": "Askim"
6 }
```

Vi kan ikke lagre alle Python-objekter i JSON-format. Objektene som vi lagde av egendefinerte klasser i blokken [2.6 Objektorientert programmering](#), må lagres med modulen `pickle`, som serialiserer til binære filer. Forsök med JSON og se hva som skjer. Eksempler på bruk av `pickle` er gitt i filene `b_2_8_3_skriv_objekter.py` og `b_2_8_3_les_objekter.py`.

For å oppsummere er JSON et format vi kan bruke når vi utveksler data mellom datasystemer (som kan kjøre forskjellig programvare på ulik maskinvare). Det er altså snakk om hvordan innholdet er formatert. Innholdet kan ligge i en fil eller som en returverdi til et API-kall. Dersom det er snakk om filer, kan disse overføres med FTP, HTTP, som vedlegg til e-poster, på minnepinner, osv.

Vedlegg [5.5 Avansert JSON](#) forklarer mer om dypere hierarki og flere dimensjoner i JSON-data. Vi vil møte på mer komplekse JSON-strukturer allerede i neste bokl hvor vi unngår problemstillingene ved å la biblioteksrutiner gjøre jobben for oss.

### Ta med minst dette

Telegrafen på 1800-tallet var en tidlig form for elektrisk kommunikasjon, men utviklingen av datakommunikasjon slik vi kjenner det i dag, begynte ikke før på 1960-tallet. **EDI** kom tidlig, etterfulgt av populære standarder som **WWW**, i form av **HTTP** og **HTML**, og **XML**. I de senere årene har **JSON** blitt et populært format for overføring av data mellom systemer som kan kjøre forskjellig programvare på ulik maskinvare. JSON har et enkelt format med objekter i krøllparenteser, som tilsvarer ordbøker (`dict`) i Python, og ordnede lister i klammparenteser.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Objektene er nøkkel-verdi par hvor nøkkelen må omgis med doble anførselstegn ("") og verdiene kan være objekter, lister, tall, strenger (tekst), true, false eller null. Vi leser inn JSON-data fra en fil ved å bruke `load`-funksjonen i Python sitt `json`-bibliotek, og vi kan skrive lister eller ordbøker til en fil i JSON-format ved å bruke `dump`-funksjonen.

### Øvingsoppgaver

#### 2.8.1

Lag et program som leser inn den vedlagte JSON-filen `p_2_8_1.json`.

- a) Lag kode som skrив ut hvor mye Elise veier:

(Plukk ut ordboken med Elise (med indeks) fra listen og `vekt` fra ordboken)

```
Elise veier 59 kg.
```

- b) Lag kode som skriver ut alle navnene med vekt og høyde:

```
Kari
vekt    : 62
høyde   : 165

Elise
vekt    : 59
høyde   : 171

Gunnar
vekt    : 84
høyde   : 187

Helge
vekt    : 76
høyde   : 178
```

#### 2.8.2

Ta utgangspunkt i oppgave 1.3.2.

- Du skal opprette en liste med `personer`
- Brukeren skal kunne taste inn `navn` og `alder`
- Når `navn` og `alder` er tastet inn, skal en ordbok med `navn` og `alder` legges til listen med `personer`
- Slik fortsetter det inntil brukeren taster retur (blankt) for å avslutte inntasting. Programmet skal skrive listen med `personer` til JSON-filen `p_2_8_2.json`, som f.eks. vist i figuren til høyre

```
> {} p_2_7_2.json > ...
[
  {
    "navn": "Kari",
    "alder": 23
  },
  {
    "navn": "Ola",
    "alder": 56
  },
  {
    "navn": "Oscar",
    "alder": 19
  }
]
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

2.8.3

Lag et program som leser inn den vedlagte JSON-filen [p\\_2\\_8\\_3.json](#).

- a) Lag kode som skriver ut bilmerkene.

```
Bilmerker:  
Toyota  
Ford  
Volkswagen  
Porsche
```

- b) Lag kode som skrive ut informasjon om den andre Ford-modellen i listen (blå Mustang)

```
En utvalgt Ford modell:  
modell : Mustang  
farge : Blå
```

- c) Lag kode som skriver ut all informasjonen (hele ordboken med alle listene med ordbøker):

```
Hele listen:  
Toyota  
    modell: Corolla  
    farge : Grønn  
  
    modell: Land Cruiser  
    farge : Hvit  
  
Ford  
    modell: Mustang  
    farge : Rød  
  
    modell: Mustang  
    farge : Blå  
  
    modell: Escort  
    farge : Hvit  
  
    modell: Explorer  
    farge : Grå
```

```
Volkswagen  
    modell: ID.4  
    farge : Grå  
  
    modell: Golf  
    farge : Gul  
  
    modell: Caravelle  
    farge : Sølv  
  
Porsche  
    modell: 911 Carrera  
    farge : Rød
```

Lag en funksjon som skriver ut alle modellene som har en bestemt farge.

```
def biler_med_farge(farge:str)->None:  
  
biler_med_farge('Rød')  
Modeller med fargen: Rød  
Ford  
    modell: Mustang  
    farge : Rød  
  
Porsche  
    modell: 911 Carrera  
    farge : Rød
```

```
Modeller med fargen: Lilla  
Ingen funnet
```

```
biler_med_farge('Lilla')
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 2.9 Reelle datasett med JSON

#### Bolkens innhold

Eurostat .....	135
Eksempeloppgave til IT-2 eksamen.....	136
Ta med minst dette.....	137
Øvingsoppgaver.....	137

#### Kompetansemål:

- velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett
- gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

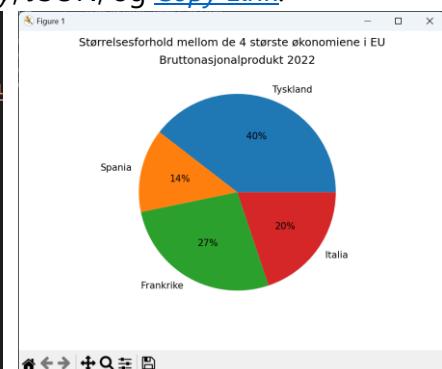
#### Eurostat

La oss si vi ønsker å lage et sektordiagram (også kjent som sirkeldiagram eller kakediagram) som viser størrelsesforholdet mellom de 4 største økonomiene i EU. Vi går til [Eurostat](#) som utfører statistisk analyse og rapportering for EU-institusjonene. Her kan vi finne [bruttonasjonalproduktene for europeiske land](#), ekskludert Storbritannia etter [Brexit](#))

Vi kan velge land, år og måleenhet. Vi velger Tyskland, Frankrike, Italia og Spania for 2022. Deretter velger vi *Current prices, million euro*. Så klikker vi på *Downloads* og velger *Options and other formats, JSON-stat (.JSON)*, *Data on this page only*, *JSON*, og [Copy Link](#).

```
from pyjstat import pyjstat
import matplotlib.pyplot as plt

url = 'https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1/data/tec00001/A.B1'
ds = pyjstat.Dataset.read(url)
df = ds.write('dataframe')
print(df.head(5)) # Vis de fem første radene
[print(x) for x in df.columns] # Vis kolonnenavn
df = df.iloc[:, [3, 5]] # Velg kolonnene med land og BNP
df.columns = ['Land', 'BNP'] # Gi kolonnene nye navn
print(df)
labels = ['Tyskland', 'Spania', 'Frankrike', 'Italia'] # Bruk norske navn på landene
plt.suptitle("Størrelsesforhold mellom de 4 største økonomiene i EU")
plt.title("Bruttonasjonalprodukt 2022")
plt.pie(df['BNP'], labels=labels, autopct='%.1f%%')
plt.show()
```



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Først lager vi en [pandas DataFrame](#) med [pyjstat](#) som i forrige eksempel. Deretter bruker vi [DataFrame.iloc](#) til å velge ut kun de kolonnene vi trenger, som er navn på land og BNP. For å lage en liste med norske navn i riktig rekkefølge, skriver vi først ut tabellen. Siden [seaborn](#) ikke tilbyr kakediagram, bruker vi [Matplotlib](#) sitt [pie](#) diagram.

### Eksempeloppgave til IT-2 eksamen

Dette er hentet fra Udir sin [eksempeloppgave](#) for Informasjonsteknologi 2, Vår 2023, Oppgave 12. Vi tar bare med a) oppgaven.

*Du skal lage et program som leser inn informasjon fra datasettet og presenterer dette i to oversikter. Du skal bruke datasettet fra forberedelsen. Hvis du ikke har forberedt dette kan du også laste ned datasettet fra forberedelsesdelen nå, [05.json](#).*

- a) *Lag et program som presenterer de tre mest brukte startlokasjonene og de tre minst brukte startlokasjonene. Presentasjonen skal også vise antall turer fra disse startlokasjonene.*

Filene er rimelig store. JSON-filen ([05.json](#)) er på 82MB og CSV-filen ([05.csv](#)) er på 32MB. Det finnes en *JSON-viewer plugin* i VS Code, men den ser ikke ut til å fungere med så store filer.

For å utforske filen, åpner vi heller CSV-filen i Excel. Husk å opprette en ny arbeidsbok, velg 'Data' og deretter 'Fra tekst/CSV' for å åpne filen.

The screenshot shows a Microsoft Excel spreadsheet with a large dataset in the background. The dataset has columns labeled A through H, containing data such as start and end times, duration, station IDs, names, descriptions, latitude, longitude, and station numbers. In the foreground, there is a PivotTable titled 'Spørrsager og tilkoblinger'. The PivotTable has two rows: 'Radetiketter' and 'Antall av start\_station\_name'. The data in the PivotTable includes:

Radetiketter	Antall av start_station_name
Alexander Kiellands Plass	2108
Ringens Park	1928
Aker Brygge	1871
Gaustad T-bane	62
Kværnerveien	59
Tordenskiolds gate	53

Det er over 160 tusen rader. Ved bruk av pivottabeller, kan vi finne ut de mest og minst populære startlokasjonene.

Siden denne blokken handler om JSON, går vi over til VS Code og Python.

Først åpner vi JSON-filen og leser innholdet inn i en [DataFrame](#). Deretter bruker vi [value\\_counts](#) på kolonnen med startlokasjoner. Vi får returnert en sortert [Series](#) med antall unike forekomster. [head\(3\)](#) henter de tre første verdiene, og [tail\(3\)](#) henter de tre siste verdiene. Vi skjøter dem med [pd.concat](#). I motsetning til [DataFrame](#), som er en todimensjonal tabell, er [Series](#) en endimensjonal tabell, som er indeksert med navn

```
import pandas as pd
from pathlib import Path

sti = Path(__file__).parent.resolve()
fil = sti.joinpath('05.json')

with open(fil) as f:
    df = pd.read_json(f)

s = df.start_station_name.value_counts() # Antall turer fra hver startlokasjon
mest_pop = s.head(3) # De tre mest populære startlokasjonene
minst_pop = s.tail(3) # De tre minst populære startlokasjonene
s = pd.concat([mest_pop, minst_pop]) # Slå sammen de to seriene

# Tabell
df_a = s.reset_index() # Lag en DataFrame av serien
df_a.columns = ['Startlokasjon', 'Antall turer'] # Gi kolonnene andre navn
print('Tre mest og minst populære startlokasjoner')
print(df_a)
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

(*start\_station\_name*).

```
print(s.iloc[2])
print(s['Aker Brygge'])
print(s.index[2])
```

1871
1871
Aker Brygge

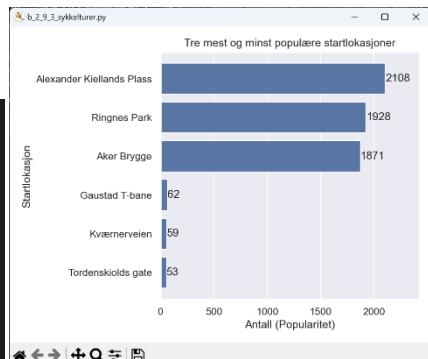
Det enkleste er å skrive ut en tabell, men for å få riktige kolonneoverskrifter, gjør vi om fra **Series** til **Dataframe** med `reset_index()`, slik at indeksen blir en kolonne.

```
# Tabell
df_a = s.reset_index() # Lag en DataFrame av serien
df_a.columns = ['Startlokasjon', 'Antall turer'] # Gi kolonnene andre navn
print('Tre mest og minst populære startlokasjoner')
print(df_a)
```

Tre mest og minst populære startlokasjoner		
	Startlokasjon	Antall turer
0	Alexander Kiellands Plass	2108
1	Ringnes Park	1928
2	Aker Brygge	1871
3	Gaustad T-bane	62
4	Kvaernerveien	59
5	Tordenskiolds gate	53

Men vi kan også skrive ut et diagram. For lange etiketter er det bedre å bruke horisontale stolper istedenfor å vippe etikettene. Dermed kan vi bruke **Startlokasjon** langs y-aksen i diagrammet. Variablen `ax` refererer til et **Axes**-objekt som inneholder de fleste elementene som inngår i diagrammet. Resten er forklaringer og formatering: Diagramtittel, verdier over stolpene, og aksetitler. `plt.tight_layout` gir nok plass til verdiene over stolpene og `plt.tight_layout` sørger for nok plass (*padding*) til akseetikettene. Vi skal lære grundigere om diagrammer i [3.2 Diagrammer med seaborn](#).

```
# Diagram
sns.set_theme() # Sett tema for plottet til standard
fig, ax = plt.subplots() # Nødvendig hvis vi vil endre tittel på vinduet
fig.canvas.manager.set_window_title(
    'b_2_9_3_sykkelturer.py') # Sett tittel på vinduet
ax = sns.barplot(data=df_a, x='Antall turer', y='Startlokasjon',
                  orient='h') # Lag horisontalt stolpediagram
ax.set(title='Tre mest og minst populære startlokasjoner') # Sett tittel på plottet
ax.set(xlabel='Antall (Popularitet)', ylabel='Startlokasjon') # Sett aksetitler
ax.bar_label(ax.containers[0]) # Legg til tallverdier på stolpene
plt.margins(0.1, 0.05) # Juster marger for å få plass til tallverdiene
plt.tight_layout() # Juster layout for å få plass til startlokasjonene
plt.show() # Vis plottet
```



## Ta med minst dette

**data.europe.eu** og **Eurostat** tilbyr åpne data innen EU, der statistiske data ofte publiseres i det flerdimensjonale **JSON-stat** formatet. Ved hjelp av **pyjstat**-biblioteket kan vi enkelt hente disse dataene inn i en **pandas DataFrame** for videre analyse og presentasjon i tabeller eller **seaborn -diagrammer**. Vi så også at pandas og seaborn var godt egnet til å løse en eksempløppgave til IT-2 eksamen. Til nå har vi kun sett eksempler på bruk av pandas og seaborn, som vi vil studere mer inngående i begynnelsen av neste runde.

## Øvingsoppgaver

2.9.1

Vi fortsetter med [eksempløppgavene](#) fra UDIR for vår 2023, Oppgave 12.

*Du skal lage et program som leser inn informasjon fra datasettet og presenterer dette i to oversikter. Du skal bruke datasettet fra forberedelsen. Hvis du ikke har forberedt dette kan du også laste ned datasettet fra forberedelsesdelen nå, [05.json](#).*

- b) Utvid programmet slik at det også presenterer et passende diagram som viser totalt antall turer fra alle startlokasjoner til sammen, per ukedag.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

2.9.2

[Kaggle](#) er en plattform for ulike datasett som også deler kode for maskinlæring og dataanalyse. I denne oppgaven skal du laste ned og analysere [News Category Dataset](#), som inneholder nyhetsartikler kategorisert etter forskjellige emner.

JSON-filen består av linjer med objekter som ikke er adskilt av kommaer og ikke er pakket inn i en liste. Du kan lese filen linje for linje direkte inn i en [pandas DataFrame](#) ved å bruke `pd.read_json()` med parameteren `lines=True`.

- a) 1) Skriv ut antall kategorier  
2) Skriv ut kategoriene  
3) Skriv ut de 3 mest og minst populære kategoriene med antall artikler
- b) Skriv ut forfatterne som har publisert mer enn 1000 artikler.  
Artikler uten oppgitt forfatter skal ikke tas med.
- c) 1) Lag et stolpediagram med antall artikler per kategori  
2) Lag et kakediagram som viser de 5 mest populære kategoriene.  
Vis den innbyrdes prosentandel av antall artikler.

Vurderingsseksemplar

## 2.10 Deepfakes

### Bolkens innhold

Bildemanipulering .....	139
Deepfakes.....	139
Anvendelser i industri og underholdning .....	140
Et tvegget sverd.....	140
God forankring i fagfornyelsen .....	141
Ta med minst dette.....	141
Øvingsoppgaver.....	142

### Kompetanse mål:

- utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger
- drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn

### Bildemanipulering

Bildemanipulering har pågått siden 1800-tallet da de første fotografiene ble tatt med kamera. Prosessen ble enklere da John og Thomas Knoll utviklet Photoshop på slutten av 1980-tallet. Vi kan justere og forbedre bilder, men vi kan også forvrenge virkeligheten. Eksempelvis kan det dreie seg om å justere farge og lyshet, kunstneriske forbedringer, fjerne kviser, gjøre tenner hvitere, øyne blåere, muskler og bryster større, endre ansiktsform, gjøre personer slankere, eller fjerne dem helt fra gruppebilder om de har falt i unåde. Kjendiser klarer ikke forhindre at det florerer med falske, pornografiske bilder av dem på nettet. Ja, vi har levd med bildemanipulasjon på godt og ondt i mange år.



### Deepfakes



Den digitale teknologien forbedrer seg hele tiden, og nå kan vi ved hjelp av kunstig intelligens manipulere video slik at vi kan "få hvem som helst til å si hva som helst". Det både ser og høres ut som en person vi vet hvem er, men det er en annen som opptrer. Ansiktet og stemmen har blitt byttet ut. Fenomenet kalles **deepfakes**, og kvaliteten blir stadig bedre. Denne [YouTube](#)-

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

[videon](#) av tidligere USAs president Barack Obama ble laget i 2018. Er dette noe vi bør bekymre oss for, eller er det gammeldags desinformasjon med en ny vri?

#### Anvendelser i industri og underholdning

Forskning og filmindustrien var først ute. Skuespillere kan gjøres yngre ([The Irishman](#)), erstattes helt med *deepfakes* av seg selv, eller digitale, ikke-eksisterende skuespillere. Filmscener kan endres uten å måtte spilles inn på nytt. Dubbing kan bli mer realistisk. [Dalí Musem](#) i Florida har brakt den avdøde, verdensberømte kunstneren tilbake til live. *Deepfakes* av historiske personer kan gjøre undervisningen i skolen mer levende og inspirerende. Kllese dem inn. Influensere kan "snakke språk" de ikke kan og nå fram til langt flere leser og tilhengere. Teknologien kan brukes til universell utforming for folk med nedsatt funksjonsevne. Kundebehandling på nettet kan blir utført av digitale personer som erstatter *chatbots*. Komikere har fått et nytt verktøy, og hvermansen kan more seg med apper som *Reface* og *Dream* (WOMBO).



I 2023 streiket folk fra filmbransjen i Hollywood, for de var bekymret for at kunstig intelligens skulle brukes til å reproduksjon av skuespilleres ytre og stemmer uten deres samtykke. Det var en historisk stor streik på vegne av fagforeningen [SAG-AFTRA](#), som representerer omrent 160,000 medlemmer. Streiken er anslått til å ha kostet USA omrent 5 milliarder dollar. Etter fire måneder ble det enighet som sikret over en milliard dollar i nye kompensasjoner og sterke beskyttelse mot AI-misbruk.

#### Et tveggje sverd

Teknologi kan brukes og misbrukes. Eksemplene er mange; alt fra atomkraft, mikrofoner og droner til kjemikalier, smarttelefoner og sosiale media. Foreløpig er mesteparten av *deepfakes* innen pornografi hvor ansiktene til de kvinnelige aktørene har blitt byttet ut, og det er ikke bare kjendiser som får gjennomgå. I tillegg til at dette er trakassering og ulovlig, åpner det samtidig muligheter for utpressing. En stund var det til og med mulig å laste ned [DeepNude](#)-appen som fikk kvinner til å fremstå nakne. Se også [artikkel i VG](#)<sup>21</sup>

Når det blir lettere å utgi seg for å være noen, blir det også lettere å lure andre. Ved hjelp av *deepfake* audio ble et firma i England [svindlet for over to millioner kroner](#) i 2019. I 2021 ble mennesker som datet på nettet angivelig [svindlet for over 5 milliarder kroner](#). Nå har bedragerne fått et enda et verktøy.

*Deepfakes* kan benyttes innen reklame, produktanmeldelser og forbrukertester. Det går an å opprette ikke-eksisterende personer, såkalte sokkedukker, som kan opptre som nettroll. Vi kan bli villedet til å tro politikere har sagt ting de ikke har sagt. Foruten alternative fakta og falske

<sup>21</sup> [Justisministeren om AI-genererte nakenbilder: – Veldig alvorlig](#)

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

nyheter kan *deepfakes* utgjøre en trussel mot et velfungerende demokrati i et samfunn med fri presse og offentlig debatt.

Kunst eller fakta? Den norske dramaserien *Atlantic Crossing*, som ble vist på NRK i 2020, ble av enkelte kritikere beskyldt for å være historieforgalskning. På samme måte som produsenten av *Antlantic Crossing* tok seg kunstneriske friheter, vil produsenter av *deepfakes* stå ovenfor etiske dilemmaer. Hvor flinke vi blir til å avsløre *deepfakes* og hvor stor innvirkning det vil få på godt og ondt, vil framtiden vise. Enn så lenge bør vi bli mer varsomme og vurdre kilder mer kritisk enn tidligere, se [\*Top 5 deepfake scams that stormed the internet this year\*](#) (2022).

#### **God forankring i fagfornyelsen**

*Deepfakes* har god forankring i **fagfornyelsen**. I vår læreplan står det under **Fagets relevans og sentrale verdier** at

*"Faget skal også bidra til at elevene reflekterer over hvordan teknologien påvirker samfunnet, og over etiske problemstillinger og demokratiske verdier som er relevante for denne påvirkningen."*

under **kjerneelementet Teknologi, individ og samfunn** som handler om

*"...hvordan informasjonsteknologi kan påvirke og endre samfunnet. Videre handler det om hvilke muligheter, utfordringer og konsekvenser bruk av informasjonsteknologi har for det enkelte individ, arbeidslivet og samfunnet. Kjernelementet handler også om etikk, personvern, universell utforming, ..."*

under det **tverrfaglige temaet Demokrati og medborgerskap** handler det om:

*"...hvordan bruk av informasjonsteknologi påvirker og utvikler samfunnet med konsekvenser for egen og andres hverdag."*

under **grunnleggende Digitale ferdigheter** står det

*"... Det innebærer videre å kritisk vurdere kilder og vise digital og etisk dømmekraft."*

under **kompetanse målene**, [KM01](#) og [KM02](#):

1. "utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger"
2. "drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn"

#### **Ta med minst dette**

*Deepfakes* er audiovisuelt innhold laget med kunstig intelligens. Det kan se og høres ut som om noen sier eller gjør ting de aldri faktisk har gjort. Dette åpner for spennende anvendelser i underholdning og utdanning. Men *deepfakes* kommer også med alvorlige utfordringer, som et kraftig verktøy til å spre desinformasjon, trakassere individer og svindle andre. Konsekvensene kan være omfattende, fra påvirkning av politiske valg til personlige overgrep

## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

og store økonomiske tap. Kildekritikk blir stadig viktigere, og Deepfakes reiser etiske dilemmaer ved å balansere ytringsfrihet og personvern mot forvrengt sannhet, økt svindelrisiko og svekket tillit i samfunnet.

### Øvingsoppgaver

#### 2.10.1

Vi skal bruke Medietilsynets [undervisningsopplegg](#) om [DEEPFAKES](#):

*"Deepfakes er en måte å bruke kunstig intelligens for å endre på video- og lydmateriale. Stemmer og ansikter ser ut som kjente mennesker i kjente omgivelser, men de er manipulert. Det kan være lett å lage og vanskelig å avsløre. Derfor er kjennskap til deepfakes viktig for å ha en god kritisk medieforståelse, og kunne avsløre falskt innhold."*

Se på videoene og svar på spørsmålene.

- a) [Hva er deepfakes, og hvorfor bør du vite om de?](#)
  - 1. Hva er forskjellen mellom en *deepfake* og en vanlig video?
  - 2. Peter Brandtzæg sier at *deepfakes* begynte med gode hensikter, men så har det blitt oppdaget av folk med onde hensikter. Hva tror du han mener med det?
  - 3. Hva tenker du er fordelene med *deepfakes*?
- b) [Slik fungerer deepfake](#)
  - 1. Hva kan være positive og negative sider ved at denne teknologien kan få folk til å snakke andre språk?
  - 2. Hvorfor tror du det lages lite *deepfake* i Norge?
  - 3. Hvordan kan *deepfakes* brukes for å lure deg?
- c) [Hvilke konsekvenser kan deepfakes ha for oss?](#)
  - 1. Hvordan kunne noen brukt *deepfakes* til å påvirke meningene dine før et valg?
  - 2. I videoen sies det at vi må jobbe med, og ikke kjempe mot den teknologiske utviklinga. Hvordan kan vi gjøre det?
  - 3. Hvordan tror du vi bruker *deepfakes* om ti år?
- b) [Slik avslører du en deepfake](#)
  - 1. Hvordan kan du finne ut hvem som er avsender av en video som dukker opp i feeden din?
  - 2. Hvordan kan vi unngå å bli lurt av falske videoer på nett?
  - 3. Hva er den slemmeste *deepfaken* du kan se for deg?

#### 2.10.2

[IT-2 Eksamens Vår 2024](#) Opgave 7 og 8

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

"Du skal svare på to oppgaver knyttet til temaet deepfakes. Du bør sette deg inn i temaet ved hjelp artiklene nedenfor. Du står også fritt til å finne andre kilder om temaet og bruke dem når du svarer på eksamensoppgavene.

- 1) [Så lett er det å gjøre Abid til Einar](#)
- 2) [Økokrim advarer mot deepfake-svindel: – Svært troverdig](#)

1)

a) Hva er deepfakes?

- en ny form for sosiale medier
- manipulerte medier
- en type virtuell virkelighet
- en populær app for ansiktsskifteffekter

b) Hvilken av følgende er en potensiell konsekvens av deepfakes?

- økt digital kompetanse blant enkeltpersoner
- økt risiko for politisk manipulering
- reduksjon av desinformasjon på Internett
- styrket tillit til video- og lydbevis

c) Hvordan kan du avsløre en godt laget deepfake-video? (Velg det alternativet som er mest riktig.)

- ved å se på videoen og avgjøre om den ser ekte ut
- ved å sjekke om videoen er laget av en profesjonell filmskaper
- ved å bruke teknikker fra kildekritikk
- ved å se om den har fått mange likes

- 2) Som student på vei ut i arbeidslivet har du blitt invitert til et jobbintervju hos en anerkjent teknologibedrift. Bedriften ønsker blant annet under intervjuet å vurdere din kompetanse til å avsløre bruk av deepfake ved et konkret svindelforsøk bedriften nettopp har vært utsatt for. En ansatt har blitt lurt til å bli med på et falskt møte med kollegaer og en kunde der bedriften har blitt svindlet for et stort pengebeløp.
- a) Hvordan kan du bidra til å avsløre tilsvarende svindelforsøk i bedriften? Hva kan bedriften gjøre for å forebygge framtidige svindelforsøk gjennom bruk av deepfake

Under intervjuet blir du også overrasket når du finner ut at bedriften bruker deepfakes for å simulere ulike arbeidssituasjoner og vurdere kandidatenes reaksjoner. Det kan være utfordrende samtaler med framtidige kollegaer og ledere, videoopptak av samhandling med kunder eller at selve kandidaten er brukt i et deepfake-scenario. Hensikten er å identifisere kandidater som kan tilpasse seg og prestere under press.

- b) Drøft konsekvenser ved bruk av deepfakes i vurdering av jobbsøkere."

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 2.11 Oppsummering

*utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger*

*Deepfakes* er redigering og manipulering av video. Ved hjelp av kunstig intelligens kan vi bytte ut personers ansikt, kropp og stemme i videoer. Vi har studert fordeler (innen filmindustrien, undervisning, underholdning, med flere) og ulemper (innen pornografi, svindel og politikk) som *deepfakes* byr på, og hvilke konsekvenser det kan medføre. Vi har også utforsket og vurdert muligheter og begrensninger med verktøy som pseudokode, flytskjema, wireframes i systemutvikling, [KM01](#).

*drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn*

*Deepfakes* forvrenger virkeligheten og kan skape tvil om hva som er sant og usant. Denne teknologien blir tilgjengelig for stadig flere, og vi vil stå ovenfor etiske dilemmaer når vi tar i bruk *deepfakes*. Hva er hensikten med å gjøre det, og hvilke konsekvenser vil det få? Hvor går grensen mellom redigering og manipulering? Hvilke regler bør samfunnet sette for bruken av *deepfakes*? [KM02](#).

*utforske og vurdere alternative løsninger for design og implementering av et program*

Vi har kodet løsninger både med kommandolinjen og GUI, [KM03](#) og [KM08](#), Tupler og ordbøker kan brukes som alternativer til lister, og vi har begynt å strukturere programmene med funksjoner, [KM07](#). Arv inngår også som en alternativ løsning, for når samme funksjon må brukes i to klasser, kan vi generalisere og legge funksjonen i en superklasse, [KM05](#). Pseudokode, flytskjema og wireframes er alternative løsninger for å beskrive og planlegge design og implementering av et program. Ved å utforske og vurdere disse verktøyene kan vi velge den mest hensiktsmessige tilnærmingen.

*anvende objektorientert modellering til å beskrive et programs struktur*

Klassediagrammer kan også inneholde klassevariabler og klassemетодer, som vises med understrek i UML. Disse er felles for alle objektene i klassen, og kan nås uten referanse til et objekt (Klassenavn.klassevariabel eller Klassenavn.klassemетодe).

Vi har også lært om tilgang til eller synlighetsmodifikatorer for attributter og metoder. **Public**, **protected** og **private** vises med henholdsvis + (plusstegn), # (nummertegn) og - (minustegn) i UML, [KM04](#).

*utvikle objektorienterte programmer med klasser, objekter, metoder og arv*

Vi har begynt å programmere våre egne klasser og objekter hvor vi har tatt i bruk attributter, metoder og arv. Vi har også lært om og programmert med nye begreper som **konstruktør**, **polyformi**, **innkapsling**, **abstraksjon**, **klassevariabler** og **klassemетодer**, [KM05](#).

Vurderingsksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

vurdere og bruke strategier for feilsøking og testing av programkode

Det finnes flere nivåer for testing av programkode hvorav **enhets-**, **integrasjons-**, **system-** og **akseptansetesting** er de vanligste. Vi har lært å automatisere enhetstesting ved å skrive testprogrammer med bruk av rammeverket **pytest** i VS Code.

Vi har også lært å ta i bruk versjonskontrollsystemet **Git**. Når vi skal rette opp i feil, kan det være nyttig å sammenlikne ulike versjoner av programkoden ved bruk av **Git**. Når vi tester programkode, må dessuten testene gjenspeile de ulike versjonene, så vi trenger også ulike versjoner av testprogrammene, [KM06](#).

generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

**Funksjoner** er den enkleste måten å lage gjenbrukbar programkode. Vi har lært flere detaljer om funksjoner; forskjellen mellom **argumenter** og **parametere**, **verdioverføring** (*passed by value*) og **referanseoverføring** (*passed by reference*) og **lokale** og **globale** variabler. Argumentene kan være posisjonsavhengige eller navngitte, og vi kan gi parameterne standardverdier og tillate et vilkårlig antall av dem. Funksjoner kan utføre en handling og returnere ingen, én eller flere **returverdier**. Når vi skal returnere flere verdier, bruker vi datatyper som lister, tupler og ordbøker.

Vi har laget objektorienterte programmer med **arv**, hvor **subklassene** (gjen)bruker attributter og metoder som er definert i **superklassen**. Vi har også **overstyrt** metoden i superklassen ved å lage en skreddersydd metode med samme navn i subklassen. [KM07](#).

vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

**Grafiske brukergrensesnitt** har blitt en standard fordi det er så brukervennlig. **GUI** er enklere å lære og bruke enn kommandolinjen. Vi har begynt å bruke **tkinter**-pakken til å programmere GUI i Python og kodet med tekstfelt, knapper, radioknapper og liste- og kombobokser. Selv om det er lettere for brukerne, er det mer arbeid for oss. Dette arbeidet kan forenkles ved bruk av biblioteker som eksempelvis **pandastable** for å vise tabeller, [KM08](#).

velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

Vi har begynt å ta i bruk **Git** til **versjonskontroll**, foreløpig med et frittstående depot for en individuell bruker. Når vi fletter sammen forskjellige versjoner fra ulike grener, kan det oppstå konflikter, som vi har lært å løse. Foruten å være et flott verktøy for versjonskontroll er Git en super samarbeidsplattform for å dele kode og jobbe som en del av et team.

Før vi sender fra oss kode til de vi samarbeider med, bør vi ha testet den ut. Vi har lært om **testdrevet utvikling** som hovedsakelig går ut på å skrive testene før vi skriver koden.

Når vi bruker **ordbøker** i stedet for **lister**, kan koden bli lettere å lese for oss og andre, for ordbøker indekseres med navn og lister indekseres med tall.

**Pseudokode**, **flytskjema** og **wireframes** er verktøy som brukes i samarbeid med andre innen systemutvikling. Det er viktig å kunne velge og bruke riktig verktøy til riktig formål, [KM09](#).

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

*gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data*

**EDI, XML** og **JSON** er standarder for utveksling av data. Vi har sett nærmere på JSON, som er utviklet for JavaScript, men kan brukes av mange andre språk, deriblant Python. JSON lagres som tekst, og formatet ligner svært på lister og ordbøker i Python. Når vi leser inn JSON-filer, blir teksten gjort om (deserialisert, *parsed*) til objekter i Python. Når vi lagrer objekter i JSON-filer, blir objektene gjort om (serialisert) til tekst på JSON-format. Dette går ikke for egendefinerte klasser. Da må vi bruke biblioteket `pickle`, [KM10](#).

*bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett*

[The official portal for European data](#) og [Eurostat](#) byr på over en million åpne datasett. Derfra har vi innhentet utvalgte data på JSON-stat format og brukt bibliotekene `pyjstat` til lesing, `pandas` til analyse og `seaborn` til grafisk presentasjon, [KM11](#).

Vurderingsseksemplar

## Runde 3

### 3.1 Jupyter, NumPy og pandas

#### Bolkens innhold

Innledning .....	147
Jupyter .....	147
NumPy .....	148
Pandas.....	149
Ta med minst dette.....	153
Øvingsoppgaver.....	153

#### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

#### Innledning

I denne bolken skal vi lære om Jupyter, NumPy og pandas. **Jupyter** er en plattform for å utvikle interaktive, nettbaserte notatbøker. Disse notatbøkene lar oss kombinere tekst, bilder og kjørbar kode i ett og samme dokument, noe som gjør det til et godt verktøy for dataanalyse og presentasjon. **NumPy** er et bibliotek for numerisk beregning i Python. Det tilbyr støtte for store, flerdimensjonale *arrays* og matriser, sammen med et omfattende utvalg av matematiske funksjoner. **Pandas** er et bibliotek for datahåndtering og analyse, bygget på toppen av NumPy og tilbyr flere høynivåfunksjoner. Det bruker enkle datastrukturer som **Series** (for endimensjonale data) og **DataFrame** (for todimensjonale data). Mens NumPy er nyttig for raske, komplekse beregninger på store datasett som hovedsakelig består av tall, tilbyr pandas verktøy for filtrering, aggregering og gruppering av data i datasett som kan inneholde forskjellige typer data, som både tall og tekst.

#### Jupyter

[Jupyter](#) er et prosjekt som utvikler plattformer for å lage interaktive, nettbaserte notatbøker som lar oss kombinere tekst, bilder og kjørbar kode i ett og samme dokument. [Jupyter Notebook](#) er gratis, åpen kildekode og et av de ledende verktøyene innen visualisering av dataanalyse. I 2022 er Jupyter utvidelsen i VS Code lastet ned over 51 millioner ganger, og i 2020 fantes det over [10 millioner Jupyter notatbøker på GitHub](#). Selv om Jupyter støtter over 70 programmeringsspråk, bruker de fleste Python.

Jupyter ble utviklet for dataanalytikere (*data scientists*). Litt enkelt forklart, kan en dataanalytiker mer om statistikk enn en informatiker og mer om informatikk enn en statistiker.



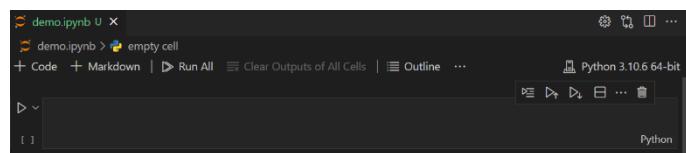
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Jupyter notatbøker egner seg også godt i klasserommet, både for lærer og elever. Det er lett å installere, enkelt å bruke og kan kjøres i en nettleser eller som en utvidelse i VS Code. Når vi er ferdige med notatboken, har vi muligheten til å eksportere resultatet i forskjellige formater som **HTML** eller **PDF**. Velg ... (More Actions ...) og **Export** fra Jupyter sin menylinje. Vi må installere ekstra programvare (LaTeX) for å kunne eksportere til en PDF-fil, og det kan dessuten by på formateringsproblemer. Derfor kan det være lettere først å eksportere til en HTML-fil, åpne denne filen i en nettleser og skrive den ut til en PDF-fil.

Installer Jupyter Notebook med `pip install jupyter` i ledetekst- eller terminalvinduet i VS Code. For å starte programmet, skriv `jupyter notebook` i ledetekstvinduet og se [Jupyter Notebook Tutorial: Introduction, Setup, and Walkthrough](#) (30 min) for å bli bedre kjent med programmet.

Jupyter fungerer ganske likt i VS Code. Installer utvidelsen, opprett en fil med filtypen **ipynb** og åpne den. En tom celle skal vises på skjermen.



Vi kan høyre-klikke i margen eller i cellen for å få fram snarveimenyen. Fem nytte kommandoer etter å ha klikket i margen er **a** (sett inn celle over), **b** (sett inn celle under), **m (markdown)**, **y (code)** og **dd** (slett celle). Vi kan også ta tak i cellen i margen og flytte den.

Klikk inni cellen for å aktivere redigering. Skriv inn Python (*Code*) eller tekst (*Markdown*) og Ctrl+↵ (Return) for utføring. *Markdown* er enkel syntaks som gjøres om til html. Last ned denne [jukselappen](#) (*cheat-sheet*), utforsk koden og kopier den inn i *Markdown* cellen og kjør den med Ctrl+↵.

<code>### Heading</code>	<b>Heading</b>
<code># H1</code>	<b>H1</b>
<code>## H2</code>	<b>H2</b>
<code>### H3</code>	<b>H3</b>
<code>### Bold</code>	<b>Bold</b>
<code>**bold text**</code>	<b>bold text</b>
<code>### Italic</code>	<b>Italic</b>
<code>*italicized text*</code>	<i>italicized text</i>
<code>### Blockquote</code>	<b>Blockquote</b>
<code>&gt;blockquote</code>	<code>blockquote</code>

## NumPy

[NumPy](#) er et bibliotek for å programmere med datababler. Det bruker sine egne objekter av typen `ndarray`, og programmene går raskere enn om vi hadde brukt `list`. Når vi finner gjennomsnittet av en million tall hundre ganger, ser vi at `NumPy` er 40 ganger raskere enn en vanlig `for`-løkke og 230 ganger raskere enn om vi hadde



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

brukt `statistics`-modulen.

```
import statistics
import numpy as np

lst = [x for x in range(100000)]
arr = np.array(lst)

def for_mean():
    total = 0
    for i in lst:
        total += i
    return total/len(lst)

def st_mean():
    return statistics.mean(lst)

def np_mean():
    return arr.mean()

[10] ✓ 0.2s
```

```
[11] ✓ 5.2s
...
for_mean: 500000
CPU times: total: 5.08 s
Wall time: 5.09 s

[12] ✓ 29.5s
...
statistics.mean: 500000
CPU times: total: 28.8 s
Wall time: 28.9 s

[13] ✓ 0.1s
...
numpy.mean: 500000
CPU times: total: 125 ms
Wall time: 119 ms
```

NumPy inneholder operasjoner og funksjoner for blant annet vektor- og matriseregning, statistiske beregninger og simuleringer. Operasjonene og metodene skiller seg fra [list](#).

Et par eksempler er vist nedenfor.

```
lst = [1,2,3]
arr = np.array(lst)

print('lst:',lst)
print('arr:',arr)
[5]
...
lst: [1, 2, 3]
arr: [1 2 3]

print('lst*2:',lst*2)
print('arr*2:',arr*2)
[6]
...
lst*2: [1, 2, 3, 1, 2, 3]
arr*2: [2 4 6]
```

```
[8]
print('[x+2 for x in lst]:', [x+2 for x in lst])
print('arr+2:',arr+2)
...
[x+2 for x in lst]: [3, 4, 5]
arr+2: [3 4 5]

[9]
lst.append(4)
print(lst)
arr = np.append(arr,4)
print(arr)
...
[1, 2, 3, 4]
[1 2 3 4]
```

Kompetansemålene går mer i retning av å analysere datasett enn vektor- og matriseregning. I neste avsnitt vil vi derfor bruke mer tid på [pandas](#)-biblioteket, som bygger på [NumPy](#). La oss likevel avslutte dette avsnittet med skalarpunkt og vektorpunkt med [NumPy](#).

```
a = np.array((1,1,0))
a
[33] ✓ 0.8s
...
array([1, 1, 0])
```

```
Python
```

```
b = np.array((0,1,1))
b
[34] ✓ 0.1s
...
array([0, 1, 1])
```

```
Python
```

```
np.dot(a,b)
[35] ✓ 0.5s
...
1
```

```
Python
```

```
np.cross(a,b)
[36] ✓ 0.6s
...
array([ 1, -1,  1])
```

```
Python
```

## Pandas

[Pandas](#) er et bibliotek for å manipulere og analysere tabeller. Det bruker egne objekter (datastrukturer), [Series](#) for endimensjonale tabeller og [DataFrame](#) for todimensjonale tabeller. En [DataFrame](#) minner mye om et regneark eller en SQL-tabell med rader og kolonner. Siden noe SQL kanskje er kjent fra IT-1, vil vi bruke denne tilnærmingen. Ofte vil vi lese inn dataene fra en fil med `pd.read_json()` eller `pd.read_csv()`, og



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

`pandas` legger dataene inn i en `DataFrame`, gjerne med kolonneoverskrifter. Vi kan også opprette en `DataFrame` fra en liste med lister. Til å begynne med er radene og kolonnene angitt med indeksnummer, men vi kan gi dem navn med `df.index` og `df.columns`. `df.info()` og `df.describe()` gir oss mer informasjon om innholdet.

The screenshot shows a Jupyter Notebook cell with the following code:

```
import pandas as pd

lst = [[1, 2, 3, 4],
       [5, 6, 7, 8],
       [9, 10, 11, 12],
       [13, 14, 15, 16]]
df = pd.DataFrame(lst)
df
```

Output:

```
[2]: ✓ 0.7s
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

Then, the code continues:

```
df.columns = ['A', 'B', 'C', 'D']
df.index = ['Øst', 'Sør', 'Vest', 'Nord']
df
```

Output:

```
[3]: ✓ 0.5s
```

	A	B	C	D
Øst	1	2	3	4
Sør	5	6	7	8
Vest	9	10	11	12
Nord	13	14	15	16

The screenshot shows two Jupyter Notebook cells. The first cell displays the output of `df.info()`:

```
[4]: ✓ 0.5s
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Øst to Nord
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   A        4 non-null      int64  
 1   B        4 non-null      int64  
 2   C        4 non-null      int64  
 3   D        4 non-null      int64  
dtypes: int64(4)
memory usage: 160.0+ bytes
```

The second cell displays the output of `df.describe()`:

```
[5]: ✓ 0.6s
```

	A	B	C	D
count	4.000000	4.000000	4.000000	4.000000
mean	7.000000	8.000000	9.000000	10.000000
std	5.163978	5.163978	5.163978	5.163978
min	1.000000	2.000000	3.000000	4.000000
25%	4.000000	5.000000	6.000000	7.000000
50%	7.000000	8.000000	9.000000	10.000000
75%	10.000000	11.000000	12.000000	13.000000
max	13.000000	14.000000	15.000000	16.000000

Vi kan plukke ut (*SQL SELECT*) kolonner ved å angi kolonnene i en liste i (dobel) hakeparenteser, og vi kan gi betingelser til hvilke rader vi vil ha med (*SQL WHERE*).

The screenshot shows three Jupyter Notebook cells demonstrating DataFrame selection:

- Cell 1: `df[['B', 'D']]` shows columns B and D.
- Cell 2: `df[df['A'] > 5]` filters rows where column A is greater than 5.
- Cell 3: `df[df['A'] > 5][['B', 'D']]` filters rows where column A is greater than 5 and then selects columns B and D.

Vi kan også plukke ut rader og kolonner med `df.iloc` (indeksnummer) og `df.loc` (navn).

The screenshot shows three Jupyter Notebook cells demonstrating DataFrame selection:

- Cell 1: `df` shows the full DataFrame.
- Cell 2: `df.iloc[[0,2],[1,3]]` selects rows 0 and 2 and columns 1 and 3.
- Cell 3: `df.loc[['Øst','Vest'],['B','D']]` selects rows 'Øst' and 'Vest' and columns 'B' and 'D'.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Vi kan legge til en ny kolonne med utregninger med normal tilordning (SQL AS) og fjerne den med `df.drop()` (SQL DROP COLUMN).

	A	B	C	D	E
Øst	1	2	3	4	2
Sør	5	6	7	8	10
Vest	9	10	11	12	18
Nord	13	14	15	16	26

	A	B	C	D
Øst	1	2	3	4
Sør	5	6	7	8
Vest	9	10	11	12
Nord	13	14	15	16

Skal vi legge til rader (SQL INSERT INTO), kan vi bruke `pd.concat`, og skal vi fjerne dem kan vi bruke `df.drop()`

	A	B	C	D
Øst	1	2	3	4
Sør	5	6	7	8
Vest	9	10	11	12
Nord	13	14	15	16
Nord-Vest	22	24	26	28

La oss lese inn et datasett for å vise sortering og gruppering:

[homes.csv](#), Home sale statistics. Fifty home sales, with selling price, asking price, living space, rooms, bedrooms, bathrooms, age, acreage, taxes. There is also an initial header line.

	Sell	List	Living	Rooms	Beds	Baths	Age	Acres	Taxes
0	142	160	28	10	5	3	60	0.28	3167
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204
4	232	240	25	8	4	3	5	2.05	3613

Først må vi rydde opp i kolonnenavnene, som er litt rotete.

```
display(df.columns)
df.columns= [x.strip().replace("'", '') for x in df.columns]
display(df.columns)

Index(['Sell', 'List', 'Living', 'Rooms', 'Beds', 'Baths', 'Age', 'Acres', 'Taxes'],
      dtype='object')

Index(['Sell', 'List', 'Living', 'Rooms', 'Beds', 'Baths', 'Age', 'Acres',
       'Taxes'],
      dtype='object')
```

Vurderingsseksempler

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

For å finne de tre dyreste hjemmene, kan vi bruke [df.sort\\_values](#) (SQL ORDER BY) og [df.head\(\)](#) (SQL LIMIT).

```
df.sort_values('Sell', ascending=False).head(3)
```

	Sell	List	Living	Rooms	Beds	Baths	Age	Acres	Taxes
28	567	625	64	11	4	4	4	0.85	12192
21	293	305	26	8	4	3	6	0.46	7088
8	271	285	30	10	5	2	30	0.53	5702

Vi kan også sortere etter flere kolonner ved å bruke lister.

```
df.sort_values(['Rooms', 'Age'], ascending=[False, True])
```

	Sell	List	Living	Rooms	Beds	Baths	Age	Acres	Taxes
43	212	230	39	12	5	3	202	4.29	3648
28	567	625	64	11	4	4	4	0.85	12192
8	271	285	30	10	5	2	30	0.53	5702
36	170	190	24	10	3	2	33	0.57	3346
38	265	270	36	10	6	3	33	1.20	5853

For å finne gjennomsnittsprisen (SQL AVG) for hjemmene gruppert etter antall rom (SQL GROUP BY) kan vi bruke [df.groupby](#) og [df.mean](#). Hvis noen av kolonnene inneholder noe annet enn numeriske verdier, må vi bruke [numeric\\_only = True](#)

```
df.groupby(df['Rooms']).mean()
```

	Sell	List	Living	Beds	Baths	Age	Acres	Taxes
<b>Rooms</b>								
5	89.000000	90.000000	10.000000	3.000000	1.000000	43.000000	0.300000	2054.000000
6	135.333333	140.000000	12.000000	3.000000	1.333333	26.000000	0.573333	2619.666667
7	139.636364	145.636364	18.181818	3.272727	1.454545	25.000000	0.782727	3132.000000
8	162.523810	169.142857	19.666667	3.904762	1.904762	24.333333	0.969524	3596.380952
9	180.875000	186.625000	22.000000	4.125000	1.875000	30.250000	1.540000	3938.500000
10	212.000000	226.250000	29.500000	4.750000	2.500000	39.000000	0.645000	4517.000000
11	567.000000	625.000000	64.000000	4.000000	4.000000	4.000000	0.850000	12192.000000
12	212.000000	230.000000	39.000000	5.000000	3.000000	202.000000	4.290000	3648.000000



pandas.DataFrame.max

pandas.DataFrame.mean

pandas.DataFrame.median

pandas.DataFrame.min

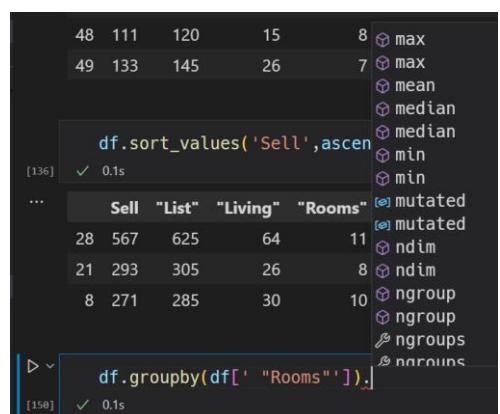
pandas.DataFrame.mode

forslagene fram.

pandas har mer å by på. Detter er forhåpentligvis tilstrekkelig informasjon for å komme i gang. Gå til [pandas.pydata.org](#), [w3schools.com](#) eller [stackoverflow.com/](#) for å finne mer informasjon, veiledning eller hjelp.

Som vi kan se i menyen til venstre når vi åpner opp dokumentasjonen for [df.mean](#), er det mye å velge mellom: max, mean, median, min, mode, osv.

IntelliSense virker også inni cellen i Jupyter utvidelsen i VS Code. Når vi taster inn punktum etter [groupby](#), kommer



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

Jupyter notatbøker består av celler som kan være enten markdown eller kode. Markdown brukes til å formatere tekst, en slags forenklet HTML-kode som er lett å lære. Kodecellene brukes for å kjøre Python-kode. Det finnes menyvalg og snarveier for det meste, samtidig som det kan være effektivt å endre til kodeceller med **y**, markdown-cell med **m**, og slette en celle med **dd** når den er markert ved å klikke i margen.

Når vi jobber med NumPy arrays, er det viktig å være klar over at matematiske operasjoner på disse tabellene oppfører seg annerledes enn på vanlige lister i Python.

Med pandas kan vi gjøre beregninger direkte på kolonner i en DataFrame, og det finnes funksjoner som `groupby` for å gruppere data, `agg` for aggregering, samt `sum`, `value_counts`, `mean` og `sort_values` som er hyppigst brukt. Pandas gjør det også enkelt å filtrere data og har også funksjoner for å lese CSV- og JSON-filer rett inn i en DataFrame ved hjelp av `read_csv()` og `read_json()`.

### Øvingsoppgaver

Du skal levere oppgaven som en Jupyter notatbok, både som **ipynb** og som **pdf**. Denne oppgaveteksten skal legges inn i *Markdown*-celler, og python koden skal legges inn i *Code*-celler. Oppgaven går ut på å manipulere og analysere datasettet nedenfor.

- datasett: <https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv>
- dokumentasjon: <https://rdrr.io/cran/reshape2/man/tips.html>

Bruk *Markdown*-celler til eventuelt å presisere svarene.

#### 3.1.1

- Vis hvor mange kolonner og rader det er i datasettet?
- Vis kolonnenavnene til datasettet.
- Vis de tre første radene
- Vis bare kolonnene *total\_bill* og *tip*
- Vis bare den 100. raden
- Vis bare kolonne *sex* og *size* for rad 10 og 11
- Vis rader hvor *total\_bill* er mindre en 8 dollar.

#### 3.1.2

- Vis de tre dyreste regningene.
- Er det flest menn eller kvinner som har betalt regningene?
- Hva er det meste som er betalt i *tip*?
- Hva er den største andelen *tip* i forhold til regningen (*total\_bill*)?
- Regn ut den gjennomsnittlige % *tip* av regningen (*total\_bill*) for menn og kvinner som har betalt den. Hvilken av de to gruppene har gjennomsnittlig tipset mest?
- Hvor mange gjester inngår totalt i datasettet?
- Hvor mange personer (hvilkem *size*) er mest vanlig (på én regning)?

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 3.2 Diagrammer med seaborn

### Bolkens innhold

Innledning .....	154
Stolpediagram .....	155
matplotlib .....	157
Styling .....	159
Kakediagram .....	160
Linjediagram .....	161
Ta med minst dette .....	161
Øvingsoppgaver .....	161

### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

### Innledning

Vi skal begrense vår analyse av data til nettopp det: å undersøke og formidle data på et ikke-teknisk språk som vanlige folk kan forstå. Det går forttere og er enklere å forstå tall når de presenteres i diagrammer. Statistisk analyse setter vi til side.

**matplotlib** Som så ofte før, når noe skal gjøres i Python, har vi mange alternativer, og det er lett å gå surr i alle uttrykkene og hvordan de skal brukes; [matplotlib](#), [pyplot](#), [plotly](#), [plot](#), [subplot](#), [figure](#), [axes](#), [axis](#), osv. Alle disse uttrykkene, bortsett fra [plotly](#), tilhører *matplotlib*, som er det eldste og mest populære biblioteket til å *plotte* diagrammer i Python.

[plotly](#) er også populært, for det lar oss lage interaktive diagrammer som vises på nettsider.



[pandas](#), som vi lærte mer om i den forrige bolken, har også sin [plot](#)-metode. *pandas* benytter *matplotlib* under panseret og gjør kodingen enklere enn om vi hadde benyttet *matplotlib* direkte.



 **seaborn** Vi har allerede brukt [seaborn](#) og skal fortsette å bruke det inntil det er noe vi ikke får til. Siden *seaborn* også bygger på *matplotlib*, kan vi benytte egenskapene og metodene i *matplotlib* når vi trenger dem. Så hvorfor bruker vi *seaborn* framfor *matplotlib*? Fordi vi får til penere diagrammer med mindre og enklere kode. Dessuten er *seaborn* tilpasset datastrukturen til *pandas* (*DataFrame*). Selv om *seaborn* er utviklet for

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

statistisk grafikk, har det også metoder for vanlige linje- og stolpediagram. Sektordiagram (kakediagram) mangler. Hvis vi allerede har dataene i en *pandas DataFrame*, kan vi da bruke [`pandas.DataFrame.plot.pie`](#) i stedet for [`matplotlib.pyplot.pie`](#).

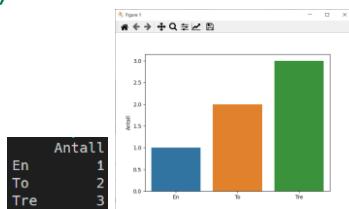
### Stolpediagram

Stolpediagram kalles også for søylediagram og blir brukt på data som er samlet i ulike kategorier, eksempelvis en bedrifts omsetning i ulike landsdeler, fortjeneste i forskjellige varegrupper eller kostnader i ulike avdelinger. Vi oppretter et stolpediagram med [`seaborn.barplot`](#) og må oppgi hva vi ønsker langs x- og y-aksen. For å vise diagrammet, må vi kalle opp [`matplotlib.pyplot.show`](#) med `plt.show()`.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.DataFrame([1,2,3],index=['En','To','Tre'],columns=['Antall'])

sns.barplot(x=df.index, y=df['Antall'])
plt.show()
```

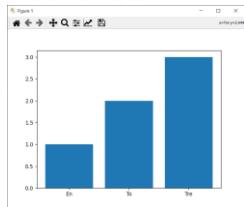


Dette hadde gått like greit med bare *matplotlib*, men det er som sagt andre grunner til at vi foretrekker *seaborn*.

```
import matplotlib.pyplot as plt

x = ['En', 'To', 'Tre']
y = [1,2,3]

plt.bar(x=x, height=y)
plt.show()
```



Vi tar for oss noen av parameterne i [`seaborn.barplot`](#) nedenfor

- data** Oppgi navn på *DataFrame*.
- x, y, hue** Oppgi kolonnenavn. Bruk *hue* for gruppert stolpediagram.
- order** Vi kan endre på rekkefølgen til stolpene.
- estimator** Vi slipper å gruppere og regne ut antall ('size'), gjennomsnitt ('mean'), minste verdi ('min'), eller lignende, for *seaborn* gjør det for oss.
- errorbar** Dersom det kommer fram en svart strek på toppen av stolpen, er dette statistisk informasjon som kan fjernes ved å sette `errorbar=None`.
- orient** Sett `orient='h'` for horisontale stolper. Det kan vi bruke når etikettenavnene langs x-asen er lange. (Husk da å bytte om x og y)
- palette** Dersom vi ønsker andre farger kan vi angi dem i en liste med navn eller fargekoder, eksempelvis 'red' eller '#FF0000'.
- dodge** står standard til *False*, som vil si at grupperte stolper står ved siden av hverandre. Dersom vi setter den til *True*, blir stolpene ikke stablet, men kommer oppå hverandre slik at noen kan skjules.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

**ax** kan vi bruke når vi ønsker flere diagrammer i samme figur. Se avsnittet om *matplotlib* nedenfor.

```
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset("tips")
df.info()
df['tip'].sum().round(0) # For kontroll av diagrammer
✓ 0.8s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   total_bill    244 non-null   float64
 1   tip          244 non-null   float64
 2   sex          244 non-null   category
 3   smoker        244 non-null   category
 4   day           244 non-null   category
 5   time          244 non-null   category
 6   size          244 non-null   int64  
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB

732.0
```

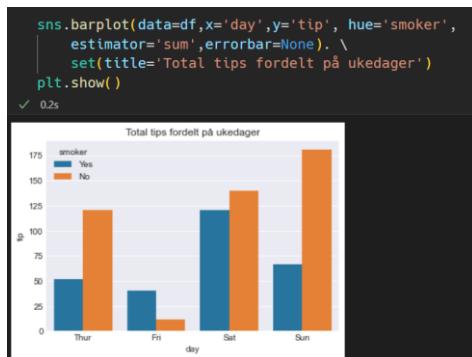
*Tips* datasettet, som vi brukte i oppgaven til den forrige blokken, følger med i *seaborn*-biblioteket og kan lastes inn med `df = sns.load_dataset("tips")`.



La oss si vi ønsker å se hvor mye tips (drikkepenger) servitøren fikk de forskjellige ukedagene. Her legger vi også til en overskrift med `.set(title=...)`.

Vi bør være forsiktig med å tolke dette dithen at folk tipser mindre på fredager. Det kan jo hende servitøren jobbet færre fredager enn lørdager og søndager, eller at det av en eller annen grunn var færre bordsetninger på fredager.

Vi har ikke sortert dataene, men kan likevel justere rekkefølgen i diagrammet med parameteren `order`.



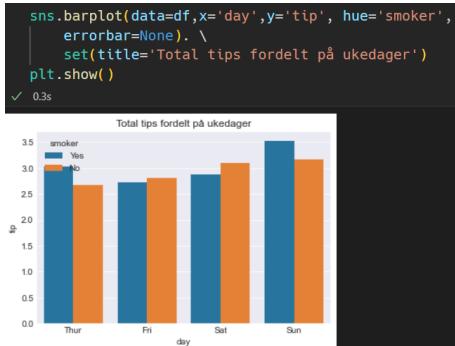
Dersom vi ønsker disse dataene gruppert på hvorvidt det var røykere ved bordet eller ikke, bruker vi parameteren `hue`.

Av samme grunn som ovenfor, er det ikke sikkert det ble tipset mindre ved bord hvor det var røykere. Det kan jo ha vært færre slike bordsetninger.

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

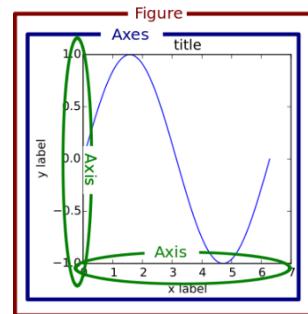


Vi kan jo se på gjennomsnittet i stedet og fjerne **estimator**, som er det sammen som `estimator= 'mean'` (*standard*). Nå ble høydeforskjellen mindre, og vi fikk omvendt resultat på torsdag, fredager og søndager. Vi må fremdeles være forsiktig med å trekke konklusjoner, for det kan være andre variabler som er avgjørende (`sex`: mann eller kvinne som betalte regningen, `time`: lunsj eller middag, `size`: antall personer ved bordet eller variabler som ikke er tatt med). Hvis vi ser et

stolpediagram hvor IQ-en stiger med skonummer, er det neppe fordi folk med store sko er smartere, men heller at de har større føtter enn barn. Fargeforklaring er nå uheldig plassert, men den kan vi endre på.

### matplotlib

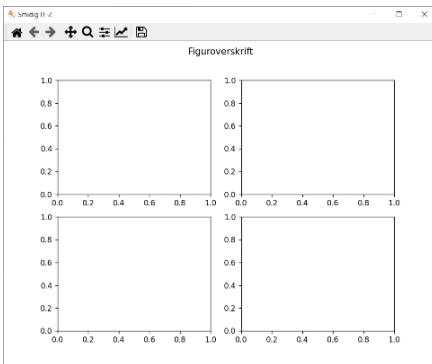
Det viktigste objektet i *matplotlib* er kanskje instanser av klassen [Axes](#) hvor plotting foregår. Et eller flere diagrammer kan plasseres på lerretet som er av klassen [Figure](#). Vi får tak i [Axis](#)-objektene (x-aksen og y-aksen) gjennom Axes-objektet.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
fig, ax = plt.subplots(2,2)
print(type(fig))
print(fig)
print(type(ax))
print(ax[0,0].xaxis) # plot øverst til venstre
print(ax[0,0].yaxis) # plot øverst til venstre
fig.set_size_inches(8,6)
fig.suptitle('Figuroverskrift')
fig.canvas.manager.set_window_title ('Smidig IT-2')
plt.show()
```



```
<class 'matplotlib.figure.Figure'>
Figure(640x480)
<class 'numpy.ndarray'>
[[<AxesSubplot::> <AxesSubplot::>]
 [<AxesSubplot::> <AxesSubplot::>]]
XAxis(80.0,254.4)
YAxis(80.0,254.4)
```

Figure og Axes oppretter vi med funksjonen `plt.subplots()` og kan blant annet bestemme figurens størrelse, overskrift og vinduets tittellinje. Når vi oppretter et diagram i seaborn, opprettes samtidig et Figure og et Axes objekt. Vi kan få tak i referansen til objektene med `plt.gcf()` (*get current figure*) og `plt.gca()` (*get current axes*), eller vi kan skrive `ax = sns.barplot(...)` og `fig = ax.figure`. **OBS!** `ax = sns.barplot(...)` fungerer ikke dersom vi avslutter med andre metoder, som f.eks `.set (title=....)`.

```
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset("tips")

fig, ax = plt.subplots(1,3)
fig.set_size_inches(12,5)
fig.suptitle('Stolpediagram')
fig.canvas.manager.set_window_title ('Smidig IT-2')
fig.tight_layout(pad=3)

sns.barplot(data=df,x='day',y='tip',
            estimator='sum',errorbar=None, ax=ax[0]). \
            set(title='Total tips fordelt på ukedager')

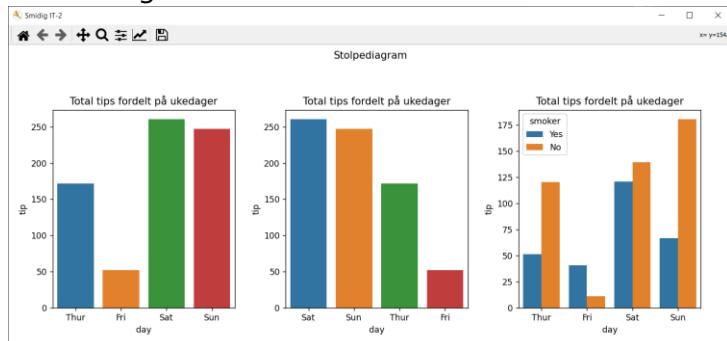
sns.barplot(data=df,x='day',y='tip',
            order=['Sat','Sun','Thur','Fri'],
            estimator='sum',errorbar=None, ax=ax[1]). \
            set(title='Total tips fordelt på ukedager')

sns.barplot(data=df,x='day',y='tip',
            hue='smoker',
            estimator='sum',errorbar=None,ax=ax[2]). \
            set(title='Total tips fordelt på ukedager')

plt.show()
```

La oss legge de tre første stolpediagrammene i denne bolken etter hverandre inn i samme figur.

`fig.tight_layout(pad=3)` justerer diagrammene så de passer fint inn i figurområdet.



Vi bruker `ax`-parametren i `barplot`, som vi skrev om innledningsvis, og putter hvert diagram inn i respektive Axes-objekt.

```
cp = sns.catplot(data=df, x='day',y='tip', col='smoker', errorbar=None, kind='bar')
cp.figure.suptitle('Total tips fordelt på ukedager')
cp.figure.tight_layout()
plt.show()
```

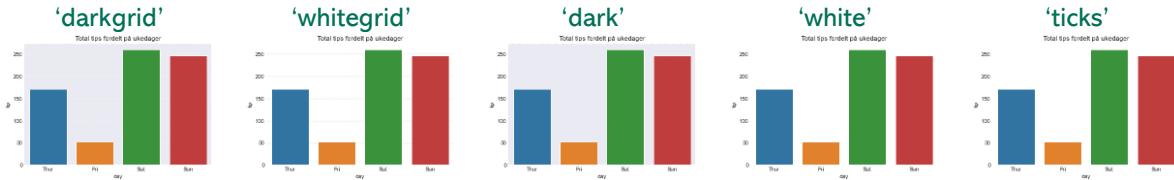
Det finnes også overordnede plot (*Figure-level*) som lar oss plotte flere like diagrammer (*Axes-level*) inn i samme figur. Her deler vi det siste, grupperte diagrammet på forrige side inn i to diagrammer ved siden av hverandre med `catplot` (bruk `row='smoker'` for å få dem over hverandre).

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Styling

En ting er hva vi **kan** gjøre, og en annen ting er hva vi **bør** gjøre. La oss først se litt på hva vi kan gjøre. `sns.set_theme()` tilbakestiller de visuelle parameterne til *seaborns* standardverdier. Vi kan bruke et av de innebygde stilene med `sns.set_style(<stil>)`



Vi kan også justere størrelse på elementer i diagrammene med `sns.set_context(<kontekst>)`



Vi kan også sende med en ordbok i `sns.set_style()` for å overstyre de innebygde stilene. Vi må da begrense oss til elementer som er med i `style` definisjonen i *seaborn*. Disse elementene, eksempelvis `xtick.color`, kan vi få listet ut med `sns.axes_style()`. Dersom elementet ikke er med i denne listen, kan vi dykke ned i *matplotlib* og lete i `rcParams`, hvor vi eksempelvis finner `xtick.labelsize` som vi kan endre med `ax.xaxis.set_tick_params()`. Det finnes mange `setter`-funksjoner i *matplotlib*, eksempelvis `ax.set_xlabel()`, som vi kan bruke til å skreddersy navnet på x-aksen.

```
import matplotlib.font_manager as fm
fm.findSystemFonts()
```

returnerer en liste med valgmulighetene for skriftyper.

Standardpaletten i *seaborn* er den kvantitative fargepaletten `tab10` i *matplotlib*, men med noe mindre intensitet. Når det gjelder farger, er det mye å velge mellom, og vi viser til siden [Choosing color palettes](#). Vi kan lage vår egen palett med et `list`-objekt med X11 [fargenavn](#) (`palette_1`) eller HEX fargekoder (`palette_2`), vi kan få fargene fordelt med like mellomrom utover fargespekteret (`palette_3`) eller vi kan velge en [\(color\)Brewer](#)-palett (`palette_4`). Hvis vi jobber i Jupyter, kan vi til og med utforske Brewer-palettene med `sns.choose_colorbrewer_palette()` før vi bestemmer oss.



Tableau Palette
#1f77b4
#ff7f0e
#2ca02c
#d62728
#9467bd
#9e9ac8
#ffbb78
#bcbd22
#17becf

Hvis vi ønsker å skrive ut hex-kodene til fargene i en palett, kan vi bruke `as_hex()`.

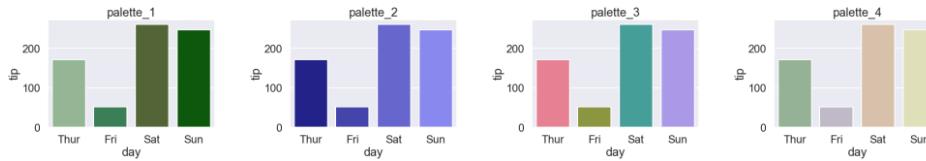
```
palette_4 = sns.color_palette('Accent', 8, 0.5)
print(palette_4.as_hex())
['#92b692', '#c0b8ca', '#dfc1a4', '#e6e6b2', '#567092', '#b43e7c', '#956341', '#666666']
```

# Smidig IT-2

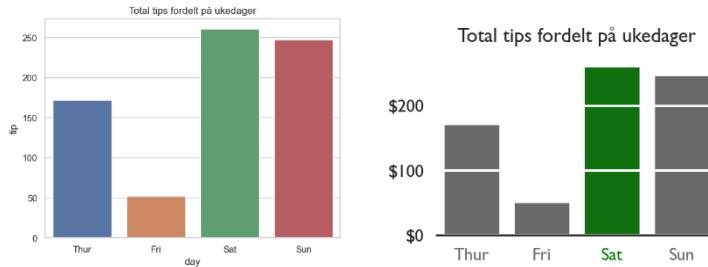
## for eksklusiv bruk ved <navn på skole>

Her er eksempler på de fire mulighetene til å lage paletter som beskrevet på forrige side.

```
sns.set_theme() # reset
sns.set_context('poster')
# palette_1 = ['DarkSeaGreen', 'Seagreen', 'DarkOliveGreen', 'DarkGreen']
# palette_2 = ['#111199', '#3333BB', '#5555DD', '#7777FF']
# palette_3 = sns.color_palette('husl', 4)
palette_4 = sns.color_palette('Accent', 8, 0.5)
sns.set_palette(palette_4)
sns.barplot(data=df, x='day', y='tip',
            estimator='sum', errorbar=None). \
            set(title="palette_4")
```



Til slutt litt om hva vi bør gjøre. Hvordan bør det visuelle budskapet formidles? Grafisk design handler om harmoni, balanse, rytme, at noe er felles, likevekt, retning, avstand, intervaller, om linjer, farger, former, struktur, tomrom, osv. Når vi skal vise data, kan vi velge mellom setninger, tabeller og grafer, gjerne en kombinasjon av disse.



[Edward Tufte](#) er spesialist på informasjonsvisualisering. Her har vi forsøkt å følge hans råd om å fjerne "diagramsøppel" (*chartjunk*) for å forbedre stolpediagrammet. Se *seaborn API Reference* og *matplotlib Default values and styling* for valg av parametere. Grønnfargen er tatt med mest for å vise hvordan vi kan styre detaljer. Tufte selv ville sikkert brukt en tabell fordi "*tables are preferable to graphics for many small data sets*".

Fredager	\$52
Torsdager	\$172
Søndager	\$247
Lørdager	\$260

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator

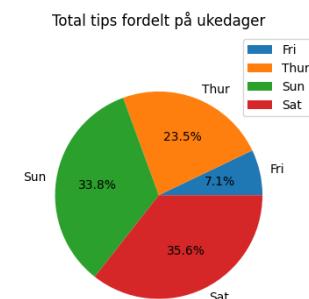
rc={'font.family':'Gill Sans MT',
    'axes.titlesize':25,
    'axes.titlepad':20,
    'xtick.labelcolor':'dimgrey'}
palette =[ 'dimgrey','dimgrey','#008000','dimgrey']
sns.set_theme(style = 'white', context='poster',
               rc=rc, palette=palette)
ax = plt.gca()
ax.yaxis.set_major_locator(MultipleLocator(100))
ax.yaxis.set_major_formatter('${x:1.0f}')
ax.yaxis.grid(color= 'white', linewidth=3)
ax.get_xticklabels()[2].set_color('#008000')
sns.despine(bottom = False, left = True)

df = sns.load_dataset("tips")
sns.barplot(data=df, x='day', y='tip',
            estimator='sum', errorbar=None, zorder=0). \
            set(title="Total tips fordelt på ukedager",
                 xlabel='', ylabel='')

plt.gcf().tight_layout()
plt.show()
```

## Kakediagram

Kakediagram, som også kalles sektordiagram, brukes til å vise hvor store de enkelte verdiene er i forhold til helheten, eksempelvis markedsandeler eller kostnadsfordeling. Edward Tufte er ikke spesielt begeistret for kakediagram: "*A table is nearly always better than a dumb pie chart; the only worse design than a pie chart is several of them*". Som vi skrev innledningsvis, er ikke kakediagram med i *seaborn*, men vi kan bruke *pandas*, se [pandas.DataFrame.plot.pie](#). Vi har krympt sirkelen med *radius*, for den ble dekket av forklaringen, og vi viser prosentene med *autopct*.



```
data = df.groupby('day').sum().sort_values('tip')
data.plot.pie(y='tip',
              radius=0.8, autopct = '%.0f%%'). \
              set(title='Total tips fordelt på ukedager',
                  ylabel = '')
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Linjediagram

Linjediagram bruker vi når noe endrer seg over tid. Her bruker vi datasettet [women\\_in\\_the\\_house\\_of\\_representatives.csv](#) og [seaborn.lineplot](#). Innholdet i `years` er en `str` på formen 1917-1919. Først plukker vi ut de 4 første karakterene med `.str[0:4]`. Så gjør vi de om til `int` med `.astype(int)`. Til slutt gjør vi linjen tykkere med `linewidth` og setter vår egen farge med `color`.



```
with open('women_in_the_house_of_representatives.csv', encoding='utf-8') as fil:  
    df = pd.read_csv(fil)  
df['years'] = df['years'].str[0:4].astype(int)  
sns.lineplot(data=df, x='years', y='democratic',  
             linewidth=3, color='mediumblue'). \  
set(title='Antall kvinnelige, demokratiske politikere i representantenes hus i USA')
```

### Ta med minst dette

Når vi har dataene våre i en `DataFrame`, er det lett å lage diagrammer med `seaborn`. Alt vi trenger å gjøre er å importere `seaborn` og `matplotlib`, og deretter bruke funksjoner som `sns.barplot()` for stolpediagrammer og `sns.lineplot()` for linjediagrammer. Vi kan spesifisere kolonner for x- og y-aksen, samt bruke parametere som `hue` for gruppering og `palette` for fargevalg. I stedet for å bruke pandas til å gruppere og regne ut antall (`size`), gjennomsnitt (`mean`), minste verdi (`min`) eller lignende, kan vi bruke `seaborns estimator-parameter` som gjør jobben for oss. Hvis vi trenger mer tilpasning, kan vi kombinere `seaborn` med `matplotlib`. Skal vi vise et kakediagram (sektordiagram), kan vi bruke enten pandas med `DataFrame['kolonnenavn'].plot.pie()` eller `matplotlib` med `plt.pie(DataFrame['kolonnenavn'])`.

### Øvingsoppgaver

Bruk det vedlagte datasettet [women\\_in\\_the\\_house\\_of\\_representatives.csv](#).

#### 3.2.1

Lag et stolpediagram over antall kvinnelige republikanere i representantenes hus i USA fra og med 1991 til og med 1995.

**Tips:** Bruk `df[df['years'].between(1991,1995)]` etter at `years` er gjort om til `int` som vist i det siste avsnittet Linjediagram i 3.2 Diagrammer med `seaborn`.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

3.2.2

- a) Finn året da forholdet mellom kvinnelige demokrater og republikanere var størst.
- b) Lag et kakediagram for kvinnelige demokrater og republikanere dette året.  
Ikke glem tittel.

Alt 1. Legg verdiene rett inn i *matplotlibs pie* funksjon (*hardcoding*):

```
fig, ax = plt.subplots()  
ax.pie([verdi_demokrater, verdi_republikanere], labels=['Demokrater','Republikanere'],  
autopct='%.1f%%')
```

Alt 2. Bruk [pandas.melt](#) for å slå sammen kolonnene `republican` og `democratic` til én kolonne som kan brukes som `labels` i `df.plot.pie`. Se [Reshaping by melt](#) og [Omformede tabeller med melt og pivot](#). 3.8 Fortsettelse av pandas

3.2.3

Lag et linjediagram over antall kvinnelige republikanere i representantenes hus i USA fra og med 1929 til og med 1937.

Vurderingsseksemplar

### 3.3 Sett og boolsk algebra

#### Bolkens innhold

Sett.....	163
Boolsk algebra.....	164
Relasjonsoperatorer.....	166
If-setninger.....	166
Sammensatte betingelser.....	167
Logikk.....	167
Ta med minst dette.....	168
Øvingsoppgaver.....	168

#### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program

#### Sett

Python har grunnleggende, innebygde datastrukturer som lar oss organisere data i samlinger (*collections*). Blant disse sammensatte datatypene finner vi [list](#), [tuple](#), [dictionary](#) og [set](#), som også blir kalt beholdere (*containers*). På lik linje med **datatypene** [int](#), [float](#), [str](#) og [bool](#) har **datastrukturene** tilhørende metoder som lar oss utføre operasjoner på samlingene med data.

Tidligere har vi behandlet [list](#) ([1.5 Lister](#)), [tuple](#) og [dictionary](#) ([2.2 Tupler og ordbøker](#)). Nå skal vi studere [set](#). Et sett er en uordnet samling uten duplikater. Vi kan ikke endre på verdiene eller hente ut verdier med en indeks, men det er mulig å legge til og trekke fra verdier og sjekke hvorvidt verdier er med i mengden eller ikke. Vi kan lage en sortert liste av et sett med [liste = sorted\(sett\)](#). Sett egner seg til å fjerne duplikater fra lister, og vi kan også utføre operasjoner som union og snitt fra mengdelæren.

```
sett_1 = {1,2,3,3,2,1}      # Fjerner duplikater
print(type(sett_1),sett_1)
liste = [1,2,3,3,2,1]
print(type(liste),liste)
sett_2 = set(liste)        # Fjerner duplikater
print(type(sett_2),sett_2)
✓ 0.7s
<class 'set'> {1, 2, 3}
<class 'list'> [1, 2, 3, 3, 2, 1]
<class 'set'> {1, 2, 3}
```

Vi oppretter et sett med krøllparenteser `{verdi_1, verdi_2, verdi_3,...}` eller den innebygde funksjonen [set](#).

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

```
venner = {'Per', 'Pål'}
print(venner)
print('Per' in venner)    # Per er med
print('Espen' in venner) # Espen er ikke med
venner.add('Espen')      # Legg til Espen
print(venner)
venner.discard('Pål')    # Fjern Pål
print(venner)
✓ 0.4s
{'Per', 'Pål'}
True
False
{'Espen', 'Per', 'Pål'}
{'Espen', 'Per'}
```

Foruten `discard` kan vi også fjerne et element med `remove`, men da får vi feilmelding om elementet ikke finnes i settet. Vi kan også bruke `pop` og ta vare på elementet, men vi har ingen kontroll på hvilket element som hentes/fjernes.

```
arilds_venner = {'Per', 'Pål', 'Espen', 'Tuva', 'Ingrid'}
almas_venner = {'Lise', 'Kristine', 'Tuva', 'Pål', 'Erik'}
samtlige_venner = arilds_venner.union(almas_venner)
print(samtlige_venner)
samtlige_venner = arilds_venner | almas_venner
print(samtlige_venner)
✓ 0.8s
{'Lise', 'Kristine', 'Espen', 'Erik', 'Per', 'Tuva', 'Ingrid', 'Pål'}
{'Lise', 'Kristine', 'Espen', 'Erik', 'Per', 'Tuva', 'Ingrid', 'Pål'}
```

For union ( $A \cup B$ ) kan vi bruke `.union` eller `|`-tegnet.

```
felles_venner = arilds_venner.intersection(almas_venner)
print(felles_venner)
felles_venner = arilds_venner & almas_venner
print(felles_venner)
✓ 0.3s
{'Pål', 'Tuva'}
{'Pål', 'Tuva'}
```

For snitt ( $A \cap B$ ) kan vi bruke `.intersection` eller `&`-tegnet.

```
venners_venner=arilds_venner.symmetric_difference(almas_venner)
print(venners_venner)
vennsters_venner=arilds_venner ^ almas_venner
print(vennsters_venner)
✓ 0.4s
{'Erik', 'Per', 'Lise', 'Kristine', 'Ingrid', 'Espen'}
{'Erik', 'Per', 'Lise', 'Kristine', 'Ingrid', 'Espen'}
```

For symmetrisk differens ( $A \Delta B$ ) kan vi bruke `.symmetric_difference` eller `^`-tegnet: Elementer som er med i A eller B, men ikke i begge.

```
almas_2grads_kontakter = arilds_venner.difference(almas_venner)
print(almas_2grads_kontakter)
almas_2grads_kontakter = arilds_venner - almas_venner
print(almas_2grads_kontakter)
✓ 0.5s
{'Per', 'Ingrid', 'Espen'}
{'Per', 'Ingrid', 'Espen'}
```

For differensmengden ( $A \setminus B$ ) kan vi bruke `.difference` eller `-` (minustegnet): Elementer som er med i A, men ikke i B.

Se [Set Types](#) for flere sett-funksjoner/operatorer.

### Boolsk algebra

Algebra forbindes ofte med ligninger hvor vi regner med tall og bokstaver. Tallene vi regner med er elementer i sett (N, Z, Q, R eller C). De grunnleggende operasjonene er addisjon, subtraksjon, multiplikasjon og divisjon. I tillegg er det regneregler som  $a + b = b + a$ ,  $(ab)c = a(bc)$  og  $a(b + c) = ab + ac$ .

Boolsk algebra er tilsvarende, men enklere. Settet er `{0, 1}` eller `{True, False}`, og det er kun tre operasjoner (`and`, `or` og `not`). Disse kan dessuten utføres med enkle elektroniske kretser. En seriekobling tilsvarer `and` og en parallellekobling tilsvarer `or`. Sammen med en `not`-krets som

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

snur (inverterer) signalet, kan vi konstruere en hvilken som helst datamaskin med kombinasjoner av disse tre kretsene. Boolsk algebra kan brukes til å forenkle logiske kretser, og vi kan bruke sammensatte logiske uttrykk i `if`- og `while`-setninger.

```
usann = (False, None, 0, '', [], (), (), {}, {})
for e in usann: print(bool(e), end=' ')
print()
sann = (True, 1, -2, 3.3, ' ', [0], {'tall':0})
for e in sann: print(bool(e), end=' ')
print()
False False False False False False False
True True True True True True
```

Objekter kan testes for sannhetsverdi, sann eller usann, `True` eller `False`, `1` eller `0`. Så det er fullt lovlig å skrive `if a:` når `a` er et objekt.

`pip install truth-table-generator`

```
import ttg
table = ttg.Truths(['p', 'q'],['p and q'])
table.as_pandas()
✓ 1.2s
   p  q  p and q
1  1  1      1
2  1  0      0
3  0  1      0
4  0  0      0
```

Operasjonen `and` er sann bare når `p` og `q` er sanne samtidig.

OG tilsvarer

```
if not p : p
else     : q
```

```
table = ttg.Truths(['p', 'q'],['p or q'])
table.as_pandas()
✓ 0.1s
   p  q  p or q
1  1  1      1
2  1  0      1
3  0  1      1
4  0  0      0
table = ttg.Truths(['p'],[not p])
table.as_pandas()
   p  not p
1  1      0
2  0      1
```

Operasjonen `or` er sann når minst én av dem er sanne.

ELLER tilsvarer

```
if not p : q
else     : p
```

Operasjonen `not p` er det motsatte av `p`.

IKKE tilsvarer

```
if not p : True
else     : False
```

Når vi regner, bruker vi pluss tegnet for `or`, gange tegnet for `and` og strek over for `not`. Det gir

$$p + 1 = 1 \quad p + 0 = p \quad p \cdot 1 = p \quad p \cdot 0 = 0$$

```
table = ttg.Truths(['p'],[p or True,p or False,'p and True', 'p and False'])
table.as_pandas()
   p  p or True  p or False  p and True  p and False
1  1          1          1          1          0
2  0          1          0          0          0
```

$$p + p = p \quad p + \bar{p} = 1 \quad p \cdot p = p \quad p \cdot \bar{p} = 0$$

```
table = ttg.Truths(['p'],[p or p,'p or not p','p and p', 'p and not p'])
table.as_pandas()
   p  p or p  p or not p  p and p  p and not p
1  1          1          1          1          0
2  0          0          1          0          0
```

La oss si vi har kommet fram til uttrykket `p and (p or q)`. Logisk algebra gir oss  
 $p(p + q) = (pp + pq) = (p + pq) = p(1 + q) = p \cdot 1 = p$

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Så vi kunne like godt brukt bare `p` i stedet for `p and (p or q)`.

table = ttg.Truths(['p','q'],['p or q', 'p and (p or q)']) table.as_pandas()				
0.7s				
p	q	p or q	p and (p or q)	
1	1	1	1	
2	1	0	1	
3	0	1	1	
4	0	0	0	

Det er et nært forhold mellom boolsk algebra og mengdelære.

Boolsk algebra {0,1}		Mengdelære (set)	
Symboler	Python	Symboler	Python
$a \wedge b$	<code>a and b</code>	$A \cap B$	<code>a &amp; b</code>
$a \vee b$	<code>a or b</code>	$A \cup B$	<code>a   b</code>

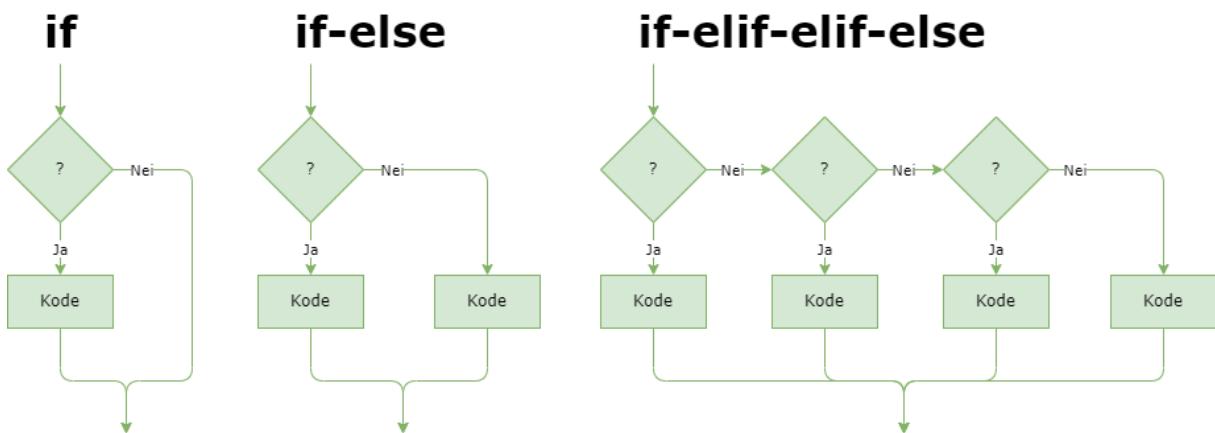
### Relasjonsoperatorer

Når vi sammenligner to verdier har vi brukt relasjonsoperatorer som vi kjenner fra matematikken:  $<$ ,  $>$ ,  $\geq$  og  $\leq$ . Skal vi sjekke om verdiene er like, bruker vi `==`, og `!=` om de er ulike. Disse operatorene blir vanligvis brukt til å sammenligne tall, men vi har også brukt dem på tekst. Resultatet blir en verdi av typen `bool`, `True` eller `False`. Som vi har sett, kan verdier også tolkes som `True` eller `False`. `None`, `0` (null), tom tekst og tommer beholdere som lister, tupler og ordbøker tolkes til `False`. Alt annet tolkes til `True` (bortsett fra en tuppel med bare én null).

Operator	Forkaring
<code>&lt;</code>	mindre enn
<code>&lt;=</code>	mindre enn eller er lik
<code>&gt;</code>	større enn
<code>\geq</code>	størren enn eller er lik
<code>==</code>	er lik
<code>!=</code>	ikke lik
<code>is</code>	objektidentitet
<code>is not</code>	negert objektidentitet

### If-setninger

Når vi må skrive forskjellige kode avhengig av bestemte betingelser, kan vi bruke if-setninger. Hvis vi skal gjøre noe bare dersom `if`-testen slår inn, klarer vi oss med `if`. Dersom vi også skal gjøre noe annet hvis testen ikke slår inn, må vi bruke `if-else`. Er det mer enn to utfall, må vi bruke en eller flere `elif` (else if). Hvis står fritt til å ha med `else` etter `elif`.



Vi kan ha `if`-setninger etter hverandre og inni hverandre (nestet). Det finnes mange variasjoner, og vi bør tenke godt gjennom logikken så det ikke blir feil. Flere alternativer kan gi samme resultatet. Noen kan være lettere å lese og forstå, mens andre kan være mer effektive. Her er noen alternativer for billettpriser (barn under 6 år gratis, ungdom under 16 år

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

kr 40, voksen kr 100 og pensjonister som har fylt 67 år kr 60). Hvilket bør vi velge?

```
if      alder < 6: pris =  0  
if 5 < alder < 16: pris =  40  
if 15 < alder < 67: pris = 100  
if 66 >= alder : pris =  60
```

```
if      alder < 6: pris =  0  
if 6 <= alder < 16: pris =  40  
if 16 <= alder < 67: pris = 100  
else                 : pris =  60
```

```
if      alder < 6: pris =  0  
elif alder < 16: pris =  40  
elif alder < 67: pris = 100  
else                 : pris =  60
```

```
if      alder < 6: pris =  0  
elif alder < 16: pris =  40  
elif alder < 67: pris = 100  
elif alder >= 67: pris =  60
```

### Sammensatte betingelser

Med boolske- og relasjonsoperatorer kan vi lage sammensatte vilkår.

```
høyde   = 181  
blå_øyne = False  
vekt    = 74  
if høyde>180 and blå_øyne and vekt<75: print('3 krav OK')  
else: print('3 krav IKKE OK')  
  
if høyde>180 or blå_øyne or vekt<75: print('minst 1 krav OK')  
else: print('minst 1 krav IKKE OK')  
  
if sum((høyde>180, blå_øyne, vekt<75))>=2: print('minst 2 krav OK')  
else: print('minst 2 krav IKKE OK')  
✓ 0s  
3 krav OK  
minst 1 krav OK  
minst 2 krav OK
```

Med flere **and** må samtlige betingelser være tilfredsstilt. Med flere **or** må minst én betingelse være tilfredsstilt. Siden en sammenligningsoperasjon returnerer **0** eller **1**, kan vi legge testene inn i en tuppel og summere resultatene. Videre kan vi kombinere flere **and** og **or** i en sammensatt test og kontrollere rekkefølgen de testene utføres med parenteser.

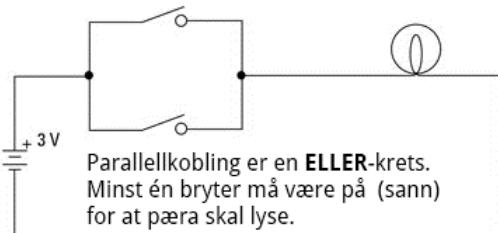
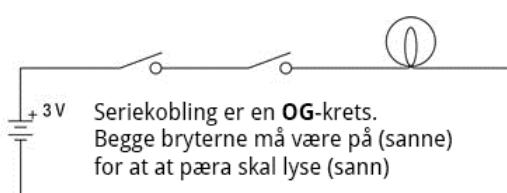
### Logikk

Datamaskiner består av logiske kretser. Hva vil det si? Logikk er jo noe vi forbinder med resonnering, argumentasjon og bevisførsel. Vi kan bruke logikk selv, for å finne beste alternativ (sunn fornuft) eller for å overbevise andre om noe. Hvis vi handler ulogisk, gjør vi noe som er ufornuftig, kanskje idiotisk eller sinnssykt. Andre behøver heller ikke ta hensyn til vår logiske argumentasjon eller utregningene til en datamaskin for den saks skyld.

I år 350 fvt. la **Aristoteles** grunnlaget for logikken som brukes i datamaskiner. Denne formallogikken beskriver regler for hvordan argumentasjonen skal bygges opp, ledd for ledd, for å komme fram til en gyldig konklusjon (Alle som levde før vår tidsregning er døde).

Aristoteles levde i år 350 fvt. Aristoteles er død). Formallogikken til Aristoteles var dominerende i over 2000 år.

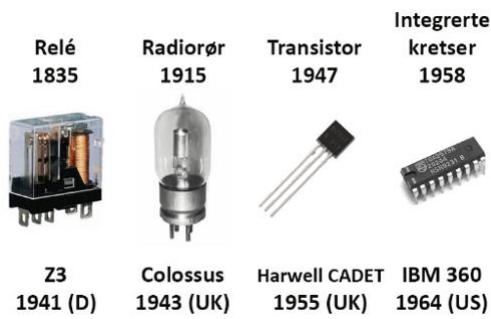
På 1600-tallet ville riktig nok **Leibniz** erstatte argumentasjon med "å regne ut hvem som har rett". Det klarte han ikke, men i sitt forsøk fant han opp totalsystemet som brukes i datamaskiner. 1 var alt og godommelig. 0 var ingenting. Først på midten av 1800-tallet klarte **Boole** å gjøre logikken om fra ord til formler. Så går det nesten enda hundre år før **Shannon**, på 1930-tallet, innså at enkle elektriske kretser kunne utføre boolsk algebra.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Alle datamaskiner kan bygges opp med kombinasjoner av de tre grunnleggende kretsene OG, ELLER og IKKE. Nå går det fort. På 1940-tallet fikk vi digitale datamaskiner basert på reléer og radiorør. På 1950-tallet ble reléene og radiorørene erstattet av transistorer. Disse ble igjen erstattet av integrerte kretser på 1960-tallet. I 2020 kunne en integrert krets bestå av milliarder transistorer som var mindre enn 10 nanometer. Til sammenligning er størrelsen på et silikonatom 0.2 nanometer. En prosessor kunne utføre to billioner instruksjoner på ett sekund.



### Ta med minst dette

Når vi bruker sett (`set`) i Python, kan vi enkelt organisere data uten duplikater og utføre operasjoner som union (`union`), snitt (`intersection`) og differens (`difference`). Boolsk algebra med operasjonene `and`, `or` og `not` hjelper oss å forenkle logiske uttrykk i programmering, spesielt i `if`-setninger og sammensatte betingelser.

### Øvingsoppgaver

Lever oppgavene som en Jupyter notatbøker (**ipynb**).

#### 3.3.1

Koden i [o\\_3\\_3\\_1.ipynb](#) skriver ut 3 Yatzy kast med 5 terninger. Oppgavene kan besvares under hverandre. Settene til **Hus** og **Fire Like** har begge to verdier. Hvordan kan vi skille disse listene fra hverandre? Se [len\(\)](#) og [list.count\(\)](#).

- Skriv ut kastene bare dersom det er **Yatzy** (Fem like terninger). Sett `antall_kast` til **5000**.
- Skriv ut kastene bare dersom det er **4 like** (Fire like terninger). Sett `antall_kast` til **100**.
- Skriv ut kastene bare dersom det er **Hus** (Kombinasjon av tre like terninger og to like terninger av en annen verdi). Sett `antall_kast` til **100**.
- Skriv ut kastene bare dersom det er **Stor straight** (Kombinasjonen 2-3-4-5-6) Sett `antall_kast` til **100**. Se [issubset\(\)](#).

```
1 import random
2 antall_terninger = 5
3 antall_kast = 3
4 for n in range(antall_kast):
5     res = []
6     for i in range(antall_terninger):
7         res.append(random.randint(1,6))
8     print(res)
9
✓ 0.8s
[4, 5, 4, 3, 2]
[2, 3, 4, 1, 3]
[2, 1, 2, 3, 3]
```

#### 3.3.2

Vi skal tegne flagg på `Canvas`-widgeten til `tkinter`. Det [norske](#) (22x16) og [svenske](#) (16x10) flagget har forskjellig proporsjoner. I denne oppgaven bruker vi de norske proporsjonene for begge flaggene og tegner det svenske flagget gult der hvor det norske er hvitt og blått.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Vi begynner med programmet **o\_3\_3\_2.ipynb** som tegner 22x16 ruter (kvadrater/rektangler). Hver rute er på 25x25 piksler så lerretet blir **550x400**. Programmet venter litt mellom hver rute for å skape en animasjonseffekt.

- **Canvas** er et lerret (*widget*) hvor vi kan tegne grafikk. Origo er i de øverste, venstre hjørnet. **x** stiger mot høyre og **y** stiger nedover.
- **Canvas.create\_rectangle(x1,y1,x2,y2, option...)** tegner et rektangel hvor (x1,y1) er posisjonen til det øvre, venstre hjørnet og (x2,y2) til det nedre, høyre hjørnet.
- **root.update()** oppdater skjermbildet.
- **time.sleep(0.01)** pauser programmet i 0,01 sekunder.

Benytt boolsk algebra til å bestemme fargen i rutene. Utgangspunktet er filen **o\_3\_2.ipynb** som farger alle rutene i det svenske flagget med blått.

- Endre blåfargen til gult på de riktige stedene i det svenske flagget
- Endre koden så det norske flagget kommer gradvis til synе rad for rad.

Flagget er delt inn i 352 ruter (22x16). Tenk at hver rute er nummerert rad for rad. Den første raden har rutene fra 0 til 21, den andre raden fra 22 til 43, osv. Bruk heltallsdivisjon **//** og modulo **%** for å finne rad og kolonne fra rutenummer. Vi kan stokke om på rutenumrene i vilkårlig rekkefølge som vist til høyre.

```
import random
ruter = [x for x in range(10)]
random.shuffle(ruter)
print(ruter)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[8, 0, 6, 5, 4, 7, 9, 3, 1, 2]
```

- Endre oppgave b) slik at det norske flagget kommer til synе rute for rute i vilkårlig rekkefølge.
- Lag tre knapper. En for hver type visning. Veksle automatiske mellom norsk og svensk flagg annenhver gang.



#### 3.3.3

a)

Hvilke alternativer gir 10% rabatt?

- medlem = 'J'** og **regning = 800**
- medlem = 'j'** og **regning = 900**
- medlem = 'J'** og **regning = 1100**
- medlem = 'j'** og **regning = 1200**

```
if medlem == 'J' or regning > 1000:
    rabatt = 10 # prosent
else:
    rabatt = 0
```

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

b)

Hvilke alternativer gir "ingen adgang"?

- A. medlem = 'ja' og vip = 'nei'
- B. medlem = 'nei' og vip = 'nei'
- C. medlem = 'ja' og vip = 'ja'
- D. medlem = 'nei' og vip = 'ja'

```
if medlem == "ja" and vip == "nei":  
    beskjed = "vent litt"  
else:  
    if medlem == "ja" and vip == "ja":  
        beskjed = "kom inn"  
    else:  
        beskjed = "ingen adgang"
```

c)

Hvilken verdi får variabelen retning dersom  
breddegrad er 31.2 og lengdegrad er -121.3

- A. 'NV'
- B. 'NØ'
- C. 'SV'
- D. 'SØ'

```
if breddegrad > 38 and lengdegrad < -134:  
    retning = 'NV'  
else:  
    if breddegrad > 38 and lengdegrad > -134:  
        retning = 'NO'  
    else:  
        if breddegrad < 38 and lengdegrad < -134:  
            retning = 'SV'  
        else:  
            if breddegrad < 38 and lengdegrad > -134:  
                retning = 'SØ'
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 3.4 Tekst, grafikk, animasjon og lyd med Pygame

#### Bolkens innhold

Pygame.....	171
Objektorientert mal for Pygame .....	172
Geometriske figurer.....	173
Tekst .....	173
Figurer og bilder .....	173
Fargebehandling .....	174
Animasjon.....	174
Lyd.....	176
Avansert animasjon.....	176
Ta med minst dette.....	178
Øvingsoppgaver.....	178

#### Kompetanse mål:

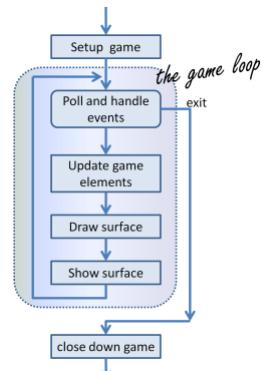
- utforske og vurdere alternative løsninger for design og implementering av et program
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

#### Pygame

Det finnes flere biblioteker for å utvikle grafiske brukergrensesnitt og animasjoner i Python. For applikasjoner som trenger klassiske GUI-komponenter, velger vi Tkinter, og Pygame for spill og animasjoner som blir enklere å programmere med bruk av *sprites*. For prosjekter som krever en blanding, kan vi enten kombinere de to eller velge basert på det mest dominerende behovet.

Installer Pygame med [pip install -U pygame](#).

I enkle programmer uten grafisk brukergrensesnitt utføres kodelinjene normalt i sekvens, én etter den andre. GUI-programmer derimot er hendelsessyrt. En hendelse oppstår når brukeren beveger musen, klikker på en knapp, trykker på en tast, og så videre. Vi programmerer ikke selve hendelsen, men heller behandlingen av den - altså hva programmet skal gjøre når hendelsen oppstår. Denne behandlingen vil imidlertid bli utforsket nærmere i bok [4.2 Hendelser, kollisjoner, tidsstyring og GUI-integrering i Pygame](#).



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Vi har brukt en modell fra boken [How to Think Like a Computer Scientist](#) (*GNU Free Documentation License*) som utgangspunkt for å lage en mal for Pygame-programmer ([b\\_3\\_4\\_01\\_pygame\\_mal.py](#)). Hovedkonseptet er at vi kjører en løkke for hvert bilde som skal vises i animasjonen. Inni denne løkken, som starter med vår egen `run`-metode, håndterer vi hendelser (`handle_events`), utfører oppdateringer (`update`), og viser innhold (`draw`) i vinduet.

### Objektorientert mal for Pygame

- Vi starter i bunnen med  
`if __name__ == "__main__":` for å utføre kodeblokken kun når filen kjøres som et skript, og ikke når den importeres som et modul i en annen fil. `app=App()` oppretter et objekt av `App`-klassen og konstruktøren `__init__` blir automatisk kjørt. Deretter setter vi i gang "The Game Loop" ved å kalle `app` sin metode `run`.
- `App`-klassen håndterer hendelser, oppdateringer og tegning.
- `__init__`-metoden (konstruktøren) initialiserer Pygame-biblioteket, setter opp skjermen, oppretter en `all_sprites` `sprite`-gruppe, og tilordner variabler for klokke og kjøring.
- En `sprite` i Pygame er et grafisk objekt som representerer en figur eller et element i spillet. `Sprites` er subklasser av klassen `Sprite` og gir oss tilgang til flere nyttige metoder for å håndtere og organisere flere `sprites` på en effektiv måte. For eksempel gjør `sprite`-grupper (`pygame.sprite.Group`) det enkelt å oppdatere, tegne og sjekke kollisjoner mellom flere `sprites` med kun noen få kommandoer.
- `handle_events`-metoden håndterer hendelser, og avslutter applikasjonen hvis `QUIT`-hendelsen utløses (når brukeren lukker vinduet).
- `update`-metoden oppdaterer alle `sprite`-objektene i gruppen.
- `draw`-metoden tømmer skjermen, tegner alle `sprite`-objektene i gruppen på skjermen og oppdaterer skjermen.
- `run`-metoden kjører hovedløkken, og håndterer hendelser, oppdateringer og tegning i en uendelig løkke, mens applikasjonen kjører.

Malen kan utvides ved å legge til flere hendelser, `sprite`-objekter og andre funksjoner og metoder etter behov.



```
1 ~ import pygame as pg
2 from pygame.locals import *
3
4 WIDTH, HEIGHT = 400, 600
5 FPS = 24
6
7
8 class App:
9     def __init__(self):
10        pg.init()
11        self.clock = pg.time.Clock()
12        self.screen = pg.display.set_mode((WIDTH, HEIGHT))
13        pg.display.set_caption("pygame mal")
14        self.running = True
15        self.all_sprites = pg.sprite.Group()
16
17    def handle_events(self):
18        for event in pg.event.get():
19            if event.type == pg.QUIT:
20                self.running = False
21
22    def update(self):
23        self.all_sprites.update()
24
25    def draw(self):
26        self.screen.fill("black")
27        self.all_sprites.draw(self.screen)
28        pg.display.update()
29
30    def run(self):
31        while self.running:
32            self.handle_events()
33            self.update()
34            self.draw()
35            self.clock.tick(FPS)
36
37
38
39 if __name__ == "__main__":
40    app = App()
41    app.run()
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

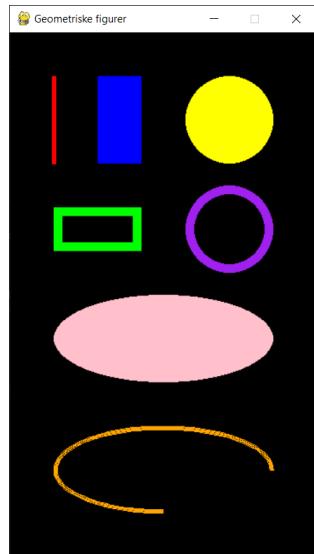
### Geometriske figurer

I [dokumentasjonen](#) finner vi `rect`, `polygon`, `circle`, `ellipse`, `arc`, `line`, `lines`, `aaline` og `aalines`. Her viser vi bare noen av disse, se filen `b_3_4_02_grafikk.py`.

```
def draw(self):
    self.screen.fill("black")
    self.all_sprites.draw(self.screen)

    # Geometriske figurer
    pg.draw.line(self.screen, 'red', (50,50), (50,150),5)
    pg.draw.rect(self.screen, 'blue', (100,50,50,100))
    pg.draw.rect(self.screen, 'green', (50,200,100,50),10)
    pg.draw.circle(self.screen, 'yellow', (250,100), 50)
    pg.draw.circle(self.screen, 'purple', (250,225), 50,10)
    pg.draw.ellipse(self.screen, 'pink', (50,300,250,100))
    pg.draw.arc(self.screen, 'orange', (50,450,250,100), 0, 1.5*3.14, 5)

    pg.display.update()
```



Linjen angis med farge, start- og slutt-punkt samt linjebredde.

Rektangelet angis med farge og (x, y, bredde, høyde) hvor x, y er posisjonen til øverste venstre hjørnet. Dersom vi ikke oppgir noen linjebredde til slutt, fylles hele rektanglet.

Det samme gjelder for sirkelen, men her angir vi sentrum og radius.

Ellipsen er som for rektanglet, og buen er inni et rektangel fra 0 til 270 grader ( $\frac{3\pi}{2}$ ).

### Tekst

For å skrive ut tekst i Pygame, kan vi benytte `SysFont`-klassen og lage en flate (`Surface`) med `render`. Deretter bruker vi metoden `blit` for å kopiere denne flaten til vindusflaten (`display`), se filen `b_3_4_03_tekst.py`.

```
def draw(self):
    self.screen.fill("black")
    self.all_sprites.draw(self.screen)

    # Tekst
    text = "Game Over"
    font = pg.font.SysFont("Arial", 50)
    text_surface = font.render(text, True, "green")
    self.screen.blit(text_surface, (WIDTH/2-100, HEIGHT/2-25))

    pg.display.update()
```

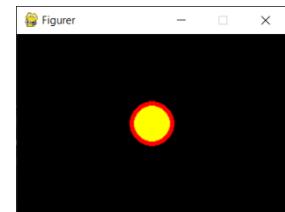


### Figurer og bilder

Når vi tegner geometriske figurer eller importerer bilder i Pygame, bruker vi ofte klasser til å representere disse objektene. Hvert objekt i en slik klasse har sin egen flate som kan tegnes på, noe som gjør det enkelt å manipulere og animere figuren. Når vi vil vise figuren på skjermen, kopierer vi objektets flate til vindusflaten ved hjelp av `blit()` eller ved å kalle `draw()` på en `sprite`-gruppe. Objektet trenger også et `rect`-attributt, som beskriver størrelse og posisjon til figuren.

I eksempelet `b_3_4_04_figur.py` tegner vi en gul sirkel med rød kant på en flate som er akkurat stor nok, og plasserer figuren midt på skjermen når vi oppretter objektet. Objektet blir

```
class Ball(pg.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pg.Surface((50, 50))
        self.rect = self.image.get_rect()
        pg.draw.circle(self.image, "yellow", (25, 25), 25)
        pg.draw.circle(self.image, "red", (25, 25), 25, 5)
        self.rect.x = (WIDTH-self.rect.width)/2
        self.rect.y = (HEIGHT-self.rect.height)/2
```



Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

opprettet og lagt til *sprite*-gruppen sist i konstruktøren til *App*-klassen, og resten går av seg selv.

Når vi importerer et bilde, som i [b\\_3\\_4\\_05\\_bilde.py](#), bruker vi `pygame.image.load()` til å laste inn bildefilen og lagre den på en *flat*. Samtidig konverterer vi pikselformatet til samme format som vindusflaten ved å kalle `convert()` på bildet. Hvis bildefilen har gjennomsiktighet, bruker vi i stedet `convert_alpha()`. I eksempelet viser vi også hvordan vi kan skalere ned bildet, men dette kan også gjøres i et bildebehandlingsprogram.

```
class Bilde(pg.sprite.Sprite):
    def __init__(self):
        super().__init__()
        # les inn bilde og omgjør til samme format som self.screen
        self.image = pg.image.load("redhatboy.png").convert_alpha()
        # skal er ned til 1/3 av opprinnelig størrelse
        self.image = pg.transform.scale(
            self.image, (self.image.get_width()//3,
                         self.image.get_height()//3))
        # sprite-objektet må ha et rect-attributt
        self.rect = self.image.get_rect()
        # plasser i midten på skjermen
        self.rect.x = (WIDTH-self.rect.width)/2
        self.rect.y = (HEIGHT-self.rect.height)/2
```

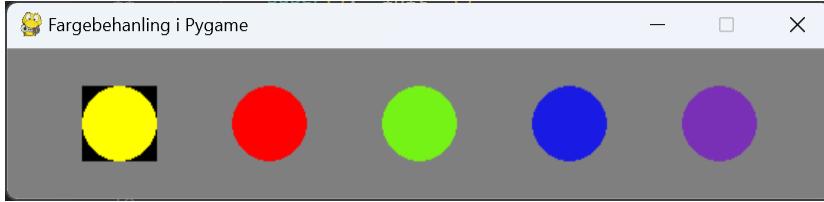
Bildet av *RedHatBoy* er hentet fra [Game Art 2D](#), som tilbyr sekvenser av bilder som kan brukes til sekundæranimasjoner.

```
class App:
    def __init__(self):
        pg.init()
        self.clock = pg.time.Clock()
        self.screen = pg.display.set_mode((WIDTH, HEIGHT))
        pg.display.set_caption("Bilde")
        self.running = True
        self.all_sprites = pg.sprite.Group()
        self.bilde = Bilde()
        self.all_sprites.add(self.bilde)
```



## Fargebehandling

Til nå har vi brukt fargenavn som '*red*', '*green*', og '*purple*' som argumenter da vi tegnet tekst og figurer. Vi kan også angi navn fra [denne listen](#) for å opprette `pygame.Color`-objekter. Alternativt kan vi definere farger ved å bruke RGBA-verdier med `pygame.Color(r, g, b, a)`, hvor **r**, **g**, og **b** representerer intensiteten for henholdsvis rødt, grønt og blått, mens **a** er alpha-verdien som bestemmer gjennomsiktigheten. Disse verdiene varierer fra 0 til 255. Med tre fargekomponenter, hver med 256 mulige verdier, kan vi skape over 16 millioner forskjellige farger. Alpha-verdien, som er valgfri, spenner fra 0 for helt gjennomsiktig til 255 for full opasitet (ugjennomsiktighet). For å tegne delvis gjennomsiktige figurer, må vi opprette flatene med SRCALPHA, `pg.Surface((bredde, høyde), SRCALPHA)`. Verdiene kan endres individuelt i etterkant, eksempelvis `color = pg.Color('blue'); color.r = 255; color.a = 64`. Dette kan brukes til myke fargeoverganger i animasjoner. Videre lar `lerp`-metoden (*linear interpolation*) oss lage overganger mellom to valgte farger. `pg.Color('yellow').lerp(pg.Color('red'), 0.7)` gir eksempelvis en oransje nyanse med 70% rødt. Se [b\\_3\\_5\\_6\\_fargebehandling.py](#).



## Animasjon

Videoer og animasjoner er bilder som vises i rask rekkefølge som vi oppfatter som bevegelse. Film og TV-sendinger viser 24 bilder i sekundet. Det vil si omtrent 42 millisekunder mellom hvert bilde. Det er ulike teknikker for å lage bildene. Et filmkamera tar opp bildene fortløpende. [Flåklypa Grand Prix](#) ble laget med såkalt *Stop Motion* teknikk. Man tar et bilde,

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

beveger dukkene litt, tar et nytt bilde, osv. Da Disney laget [Snehvit og de syv dvergene](#) i 1937, ble hvert bilde tegnet for hånd. Regn ut hvor mange bilder det blir. Nå tegnes bare nøkkelsbildene, som i en tegneserie, og deretter fyller dataprogrammer inn bildene som mangler. Denne prosessen kalles *tweening* ("inbetweening") som gir myke overganger mellom nøkkelsbildene.

I [øving 3.3.2](#) med flaggene lagde vi animasjonen i *tkinter* med `time.sleep()`. Det er ingen god løsning, for når programmet "sover", reagerer det ikke på andre hendelser. Et bedre alternativ ville vært å bruke metoden `after()`, se vedlegg [5.6 Animasjon med tkinter](#). I Pygame reguleres oppdateringshastigheten, FPS (*Frames Per Second*), med `self.clock.tick(FPS)` i den siste setningen i *The Game Loop*. Animasjonshastigheten blir bestemt av hvor hyppig vi oppdaterer og hvor langt vi flytter objektet i hvert bilde. Hvis vi flytter objektet 5 piksler 24 ganger i sekundet, blir animasjonshastigheten 120 piksler/s. Vi får samme hastighet med 10 piksler 12 ganger i sekundet. Animasjonen krever da mindre prosesseringstid, men blir mer hakkete. Ved bruk av `math.sin` og `math.cos` kan vi eksempelvis få ballen til å gå i sirkel eller følge banen til et skrått kast.

I eksempel [b\\_3\\_4\\_07\\_enkel\\_animasjon.py](#) tar vi utgangspunkt i filen [b\\_3\\_4\\_05\\_figur.py](#), der vi tegnet en gul, rund ball med rød kant. Startposisjonen har vi satt til venstre for vinduet i x-retning og midstilt i y-retning. Ballen flyttes 5 piksler (`self.dx = 5`) 24 ganger i sekundet (`FPS=24`) og beveger seg ved at vi endrer dens posisjon for hvert bilde med `self.rect.x += self.dx` i ballens `update`-metode. En positiv `dx` beveger ballen mot høyre, og tilsvarende vil en positiv `dy` bevege ballen nedover. Vi har også lagt inn en `if`-setning som flytter ballen tilbake til startposisjonen når den forsvinner ut av vinduet til høyre. Dermed glir ballen over skjermen fra venstre mot høyre gjentatte ganger. Her kunne vi kommet på flere alternativer, som å stoppe animasjon, spille av en lyd, la ballen gli tilbake den andre veien, eller utforske andre idéer. Ballen trenger ingen egen `draw`-metode, ettersom den er lagt til i `all_sprites`-gruppen, og derfor vil den bli tegnet automatisk når vi kaller `self.all_sprites.draw(self.screen)` i vår `draw`-metode som kalles fra hovedløkken.

*Tweening* kan transformere egenskaper som farge, posisjon, rotasjon og størrelse mellom nøkkelsbildene. Overgangene blir mer naturlige dersom de er ulineære. Hvis vi skal kjøre i 80 km/t fra A til B, tar det litt tid før vi kommer opp i 80 km/t, og det tar en viss tid å stanse. Dette kalles for *ease in* og *ease out* i *tweening*-sammenheng. Denne [YouTube videoen](#) sammenligner ulike *easing* typer. Det finnes *tweening* bibliotek for Python, men vi må gjøre mye selv.

`pip install pytweening --use-pep517`

```
class Ball(pg.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pg.Surface((50, 50))
        self.rect = self.image.get_rect()
        pg.draw.circle(self.image, "yellow", (25, 25), 25)
        pg.draw.circle(self.image, "red", (25, 25), 25, 5)
        self.rect.x = -self.rect.width
        self.rect.y = (HEIGHT-self.rect.height)/2
        self.dx = 5

    def update(self):
        self.rect.x += self.dx
        if self.rect.left > WIDTH:
            self.rect.right = -self.rect.width
```



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Her bruker vi `easeOutBounce` for å få ballen til å sprette et par ganger når den treffer bakken. Først må vi bestemme hvor lenge animasjonen skal være så vi vet hvor mange bilder det blir. Det gjør vi i ballens `__init__`-metode med `self.bilder = 2 * FPS`. Deretter bruker vi en teller, som argument, til å angi med prosentfaktor hvor langt vi har kommet. Returverdien er også mellom 0 og 1, men den er `tweenet`. Her lar vi dessuten ballen ligge nede i 1 sekund før den `tweenes` på nytt. Se eksempel [b\\_3\\_4\\_08\\_tweening.py](#).

```
def update(self):
    self.teller += 1
    avstand = HEIGHT - self.rect.height

    # Tween i 2 sekunder
    if self.teller <= self.bilder:
        y = avstand * tween.easeOutBounce(self.teller / self.bilder)
    # La ballen ligge nede i 1 sekund
    elif self.teller <= FPS * 3:
        y = avstand
    # Start på nytt etter 3. sekunder
    else:
        y = 0
        self.teller = 0

    self.rect.y = y
```



### Lyd

Pygame gjør det enkelt å spille av lyd i Pythonprogrammer. Vi kan velge mellom `pygame.mixer.Sound`, som laster inn små lydfiler og `pygame.mixer.Music` som strømmer audio. Vi tar utgangspunkt i [b\\_3\\_4\\_07\\_enkel\\_animasjon.py](#). Når ballen treffer kanten, spretter den tilbake, det vil si snur hastighetsretningen, og vi spiller av lyden `ping.mp3`. Tilsvarende spiller vi av lyden `pong.mp3` når ballen treffer den venstre kanten. Vi laster inn lydfilene i ballens `__init__`-metode, og spiller de av i ballens `update`-metoden. Se eksempel [b\\_4\\_3\\_09\\_lyd.py](#).

```
class Ball(pg.sprite.Sprite):
    def __init__(self):
        super().__init__()

        # Last inn lydfilene
        pg.mixer.init()
        self.PING = pg.mixer.Sound("ping.mp3")
        self.PONG = pg.mixer.Sound("pong.mp3")

        self.image = pg.Surface((50, 50))
        self.rect = self.image.get_rect()
        pg.draw.circle(self.image, "yellow", (25, 25), 25)
        pg.draw.circle(self.image, "red", (25, 25), 25, 5)
        self.rect.x = 0
        self.rect.y = (HEIGHT-self.rect.height)/2
        self.dx = 5

    def update(self):
        self.rect.x += self.dx

        # Sjekk om ballen treffer kanten
        if self.rect.left < 0 or self.rect.right > WIDTH:
            # Snu fartsretningen
            self.dx *= -1

        # Spill av lyd
        if self.dx < 0:
            self.PING.play() # Traf høyre kant
        else:
            self.PONG.play() # Traf venstre kant
```

### Avansert animasjon.

Vi avslutter med et litt mer avansert eksempel hvor 25 baller i forskjellig størrelse og farger beveger seg i ulik hastighet rundt på skjermen. Alle ballene spretter tilbake fra alle vinduskantene. En ny ball opprettes annet hvert sekund inntil det er 25 animerte baller vinduet.

App-klassens `__init__`-metode er som tidligere, med et tillegg av en `timer` variabel som holder styr på tiden. `pygame.time.get_ticks()` returnerer antall millisekunder siden `pygame.init()` ble kalt. I App-klassens `update`-metode kan vi dermed sjekke når det har gått 2 sekunder og opprette en ny ball, som vi samtidig legger inn i `all_sprites` gruppen.

```
def update(self):

    # Lag nye baller hvert 2. sekund inntil ANTALL_BALLER
    if pg.time.get_ticks()-self.timer > 2000:
        self.timer = pg.time.get_ticks()
        if len(self.all_sprites) < ANTALL_BALLER:
            self.all_sprites.add(Ball())

    self.all_sprites.update()
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

I ballen `__init__`-metode setter vi først diameteren til et vilkårlig tall mellom 25 og 100 piksler. Deretter oppretter vi en liste med tre tilfeldige tall mellom 0 og 255 til å angi RGB fargekode. Etterpå plasserer vi ballen i et vilkårlig punkt i vinduet (`self.rect.x` og `self.rect.y`) og avslutter med å gi ballen en tilfeldig hastighet mellom 2 og 8 piksler per bilde i både x- og y-retning. Legg merke til bruken av `choice` så ballen kan begynne å bevege seg mot høyre eller mot venstre og oppover eller nedover.

```
class Ball(pg.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.diameter = randint(25, 100)
        self.image = pg.Surface([self.diameter, self.diameter])
        farge = [randint(0, 255) for _ in range(3)]
        self.image.set_colorkey('black')
        radius = self.diameter//2
        pg.draw.circle(self.image, farge, (radius, radius), radius)
        self.rect = self.image.get_rect()
        self.rect.x = randint(0, WIDTH-self.diameter)
        self.rect.y = randint(0, HEIGHT-self.diameter)
        self.dx = choice((-1, 1)) * randint(2, 8)
        self.dy = choice((-1, 1)) * randint(2, 8)
```

Når ballen treffer en vinduskant, snur vi retningen til hastighetskomponentene ved å multiplisere `dx` eller `dy` med minus 1. Samtidig sørger vi for at hele ballen forblir synlig ved å sette den tilbake til kanten. Dette kunne vi ha gjort med fire `if`-setninger, men vi har valgt en mer kompakt og avansert metode ved hjelp av `min`- og `max`-funksjonene. Prøv å forklare for en medelev hvorfor vi oppnår samme resultat ved bruk av `min`- og `max`-funksjonene.

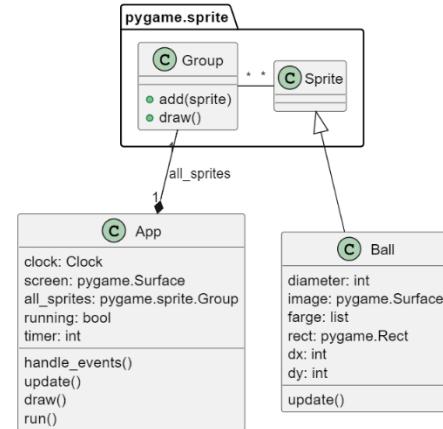
```
def update(self):
    self.rect.x += self.dx
    self.rect.y += self.dy

    # Sjekk om ballen er utenfor skjermen
    if self.rect.left < 0 or self.rect.right > WIDTH:
        self.dx *= -1
        self.rect.x = max(0, min(WIDTH - self.diameter, self.rect.x))

    if self.rect.top < 0 or self.rect.bottom > HEIGHT:
        self.dy *= -1
        self.rect.y = max(0, min(HEIGHT - self.diameter, self.rect.y))
```

Figuren til høyre viser et UML klassediagram for programmet ([b\\_3\\_4\\_10\\_avansert\\_animasjon.puml](#)).

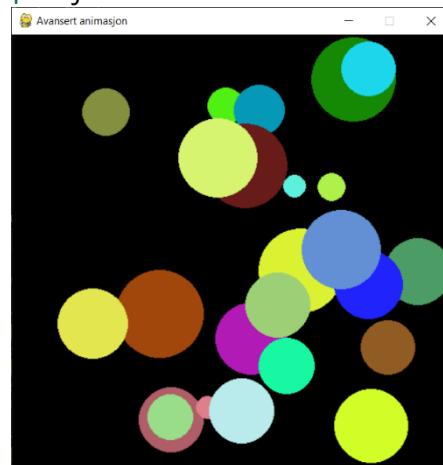
`App`-klassen, som representerer hovedapplikasjonen, er som malen vår med tillegg av variablene `timer`. Klassen har en relasjon til `Group`-klassen via variablen `all_sprites`, som gir den tilgang til medlemmer og metoder i `Group`-objekter. Dette forholdet kalles komposisjon og vises med en strek som forbinder klassene, og en fylt rombe i enden som inneholder variablen `all_sprites`. Vi skal forklare mer om [forholdet mellom klasser](#) i neste bokl 3.5 [Klassediagram](#).



`Group` og `Sprite` er klasser som tilhører `pygame.sprite`-modulen, og de vises i en egen boks som representerer denne modulen i UML-diagrammet. Vårt `Group`-objekt inneholder referanser til ballene som også er `Sprite`-objekter. Dette er også aggregering

Relasjonen mellom `ball`-objekter og `Sprite`-klassen er imidlertid arv, også kalt generalisering. Det vises med en pilspiss, som ser ut som en vanlig trekant, der spissen peker mot superklassen

Forklaring på bruk av min- og max-funksjonene: `min(WIDTH - self.diameter, self.rect.x)` sikrer at ballens x- posisjon aldri overskridet vinduets bredde minus ballens



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

diameter. Når kombinert med `max(0, ...)` sørger det også for at `x`-posisjonen aldri er mindre enn 0.

### Ta med minst dette

Pygame er et bibliotek for å lage multimedieapplikasjoner som spill, animasjoner og simuleringer. Når vi koder Pygame-programmer, er det viktig å utføre kommandoer i riktig rekkefølge. Derfor er det fordelaktig å starte med en mal som setter opp vinduet, tidsstyring og hovedløkken. Hovedløkken håndterer hendelser, oppdateringer og tegner alt på nytt. Animasjonen oppstår når objektenes posisjoner oppdateres, og de deretter tegnes på nytt, noe som får dem til å se ut som om de flytter seg. I malen kan vi justere verdier som skjermstørrelse, bakgrunnsfarge og oppdateringsfrekvens for å tilpasse programmet etter våre behov. Pygame lar oss tegne geometriske figurer, vise tekst og bilder, samt spille av lyd. Vi skal lære mer om hendelser og interaktive applikasjoner i Runde 4.

### Øvingsoppgaver

#### 3.4.1

Lag et program med følgende spesifikasjoner

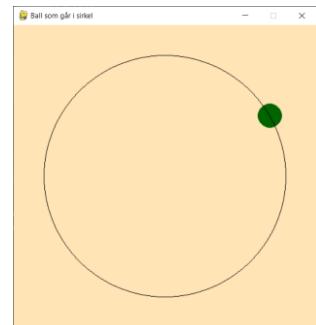
- Vinduet skal ha størrelsen 600x500
- Tegn et hus omrent tilsvarende som vist i bildet til høyre.
- TIPS:
  - Se Pygame [color list](#) for fargenavn
  - Bruk [polygon](#) til å tegne taket



#### 3.4.2

Lag et program med følgende spesifikasjoner

- Vinduet skal ha størrelsen 500x500
- En ball med diameter 40 piksler skal gå rundt i sirkel inni vinduet
- Sirkelen skal ha en diameter på 400 piksler og være sentrert i vinduet



#### 3.4.3

Ta utgangspunkt i filen `b_3_4_6_enkel_animasjon.py` og gjør følgende endringer:



- Endre bevegelsen til å bruke `easeInOutExpo`-tween for jevnere animasjon.
- La ballen animere frem og tilbake gjentatte ganger, uten avbrudd.

Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

#### 3.4.4

Lag et program med følgende spesifikasjoner

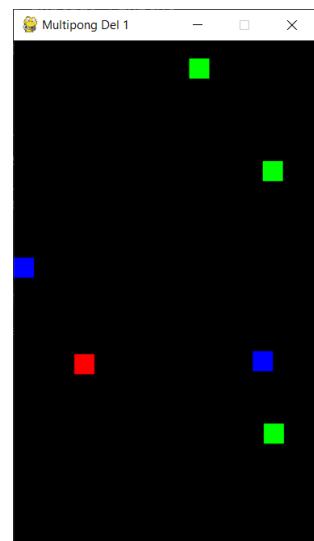
- Vinduet skal ha størrelsen 200x200.
- Programmet skal vise et nytt, tilfeldig siffer mellom 1 og 3 på skjermen hvert 1.5 sekund.
- Sifferet skal vises i 0.75 sekunder før det forsvinner fra skjermen.
- Når et nytt siffer vises, skal den tilhørende lydfilen (one.mp3, two.mp3 eller three.mp3) spilles av.



#### 3.4.5

IT-2 eksamen, eksempeloppgave 13. Se neste side.

- Foreløpig holder det med å lage baller som beveger seg nedover skjermen.
- Ballene skal endre retning når de treffer sideveggene (sprette tilbake inn i banen).
- Lag en ny ball hvert sekund.
- Padden og kollisjonene programmerer vi når vi kommer til [4.2 Hendelser, kollisjoner, tidsstyring og GUI-integrering i Pygame](#).



Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Informasjonsside om Multipong

I denne oppgaven skal du utikle en variant av klassikeren pong, nemlig multipong.  
Du bør beregne å bruke inntil 2,5 timer på denne oppgaven.

Spillet starter med følgende

- En ball ( rødig kvadrat øverst)
- En padde ( lyseblått rektangel nederst)

Når spillet starter skal ballen bevege seg nedover og mot høyre. Når den treffer høyre kant, skal ballen endre retning og bevege seg nedover mot venstre og mot padden.

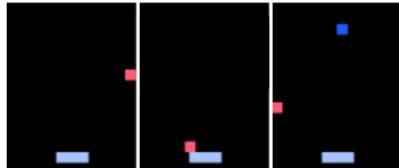
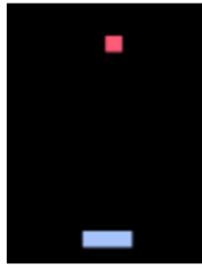
Padden skal styres med venstre og høyre piltaster.

Spiller, som styrer padden, må sørge for at ballen blokkeres, slik at den snur og beveger seg tilbake og oppover mot venstre.

Om spiller ikke greier blokkere ballen og den treffer banens nedre kant er spillet over.

Når den første ballen er blokkert - vil det komme en ny ball, og denne beveger seg enten mot venstre og nedover, eller mot høyre og nedover. Det blir altså enda en ball spilleren må blokkere. Og slik fortsetter multipong. Stadig flere baller kommer ut på banen og beveger seg nedover mot venstre eller mot høyre - og må blokkeres av spiller.

Spillet avsluttes når første ball treffer banens nedre kant.



### Oppgave 13 - Multipong: spillet og ditt oppdrag

I denne oppgaven skal du altså utvikle en versjon av Multipong

Grensesnittet skal være omtrent som vist i illustrasjonene til høyre. (Her ser du stadig nye baller komme til og må blokkeres)

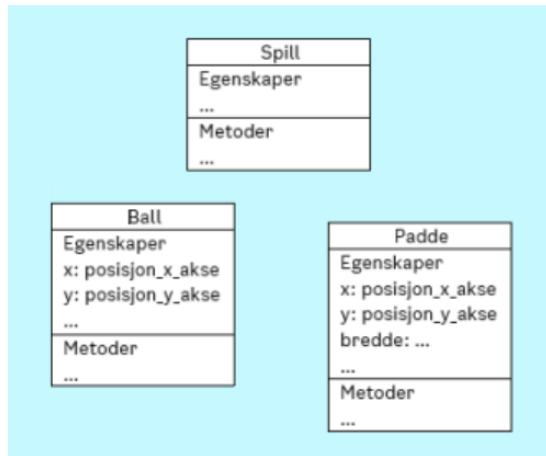
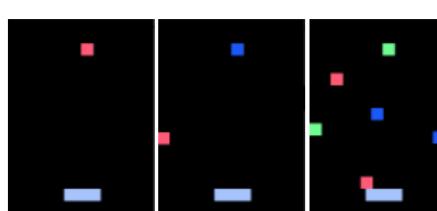
Ta utgangspunkt i klassediagrammet vist til høyre. Du må legge til egne variable og metoder i klassene i tillegg til de som allerede er skrevet inn.

Funksjonelle krav:

- Ved oppstart skal ballen bevege seg først nedover mot høyre eller nedover mot venstre
- Padden skal styres med venstre og høyre piltaст. Den skal ikke kunne flyttes utenfor banen.
- Om spiller blokkerer ballen, snur den og beveger seg opp mot høyre eller opp mot venstre.
- Når ballen så treffer banens øvre kant snur den og beveger seg igjen nedover.
- Med jevne mellomrom skal en ny ball plasseres på banen - og bevege seg nedover mot venstre høyre og spiller må forsøke å blokkere med padden
- Spillet avsluttes når spiller ikke greier blokkere en av ballen og ballen treffer banens nedre kant.

Oppdrag:

- Lag Multipong



Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 3.5 Klassediagram

### Bolkens innhold

Innledning .....	192
Kommentarer .....	193
Typeantegnelser.....	194
dir, help, pydoc og IntelliSense .....	194
Docstring .....	195
Dokumentasjonsverktøy .....	196
Ta med minst dette.....	197
Øvingsoppgaver.....	198

### Kompetanse mål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- anvende objektorientert modellering til å beskrive et programs struktur
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

### UML

I blokk [1.8 Modellering](#) ble vi kjent med UML og klassediagram, noe vi jobbet mer med i blokk [2.6 Objektorientert Programmering](#). Det finnes totalt [14 diagramtyper](#) fordelt på [strukturelle og adferdsbaserte diagrammer](#). Vi skal fokusere på de tre mest brukte diagramtypene: Klassediagram (*class diagrams*) , brukstilfellediagram (*use case diagrams*) og sekvensdiagram (*sequence diagrams*). I denne bolken vil vi fullføre klassediagram, mens brukstilfellediagram og sekvensdiagram vil vi ta for oss i neste runde.

### Klassediagram

Klassediagram kalles også en domenemodell, og det representerer det aktuelle problemområdet som skal løses. En viktig del av arbeidet er å dele opp problemet i klasser. Klassene representerer vanligvis virkelige objekter. Hvordan kommer vi opp med klassene? Det finnes flere metoder, og en mulighet er å bruke eller modifisere en eksisterende modell. En annen strategi er substantivmetoden.

### Substantivmetoden

Substantivmetoden innebærer å identifisere alle substantiver i kravspesifikasjonen og vurdere dem som potensielle klasser eller attributter. Nedfor er informasjonssiden om Multipong, som var oppgave 13 i [eksempløppgaven](#) i den nye IT-2 eksamen.

**Spillet** starter med følgende

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

- En ball (rødlig kvadrat øverst)
- En padde (lyseblått rektangel nederst)

Når spillet starter, skal ballen bevege seg nedover og mot høyre. Når den treffer høyre kant, skal ballen endre retning og bevege seg nedover mot venstre og mot padden.

Padden skal styres med venstre og høyre piltaster.

Spiller, som styrer padden, må sørge for at ballen blokkeres, slik at den snur og beveger seg tilbake og oppover mot venstre.

Om spiller ikke greier blokkere ballen og den treffer banens nedre kant er spillet over.

Når den første ballen er blokkert - vil det komme en ny ball, og denne beveger seg enten mot venstre og nedover, eller mot høyre og nedover. Det blir altså enda en ball spilleren må blokkere. Og slik fortsetter multipong. Stadig flere baller kommer ut på banen og beveger seg nedover mot venstre eller mot høyre - og må blokkeres av spiller.

Spillet avsluttes når første ball treffer banens nedre kant.

Kandidater er altså spill (4), ball (10), kvadrat (1), padde (4), rektangel (1), kant (3), retning (1), piltast (1), spiller (4), bane (3) og multipong (1). Ball, padde, spill og spiller forekommer mest hyppig. Attributter er vanligvis primitive datatyper som `int`, `float`, `str` eller `bool`. Hvis et begrep består av flere deler, er det sannsynligvis en klasse. En kant kan representeres som et attributt (én verdi). Banen har fire kanter, så vi kunne ha et Bane-objekt med fire kant-attributter, eller vi kan la de være en del av Spill-klassen. Multipong er det samme som spill, og piltastene er koblet til hendelser. Kvadrat og rektangel er naturlige deler av henholdsvis Ball- og Padde-klassen. Så har vi igjen retning, som kan ha to verdier. Vi kunne ha laget en egen klasse for retning, men det kan også være to attributter i Ball-klassen. Trenger vi Spiller-klassen? Slik oppgaven er formulert styrer spilleren padden, og det fanger vi opp gjennom hendelser. Hvis vi derimot ønsker å lagre informasjon om spillernes navn, oppnådd poengsum, datoer, osv., ville det være naturlig å inkludere en Spiller-klasse. Dermed står vi igjen med Spill-, Ball- og Padde-klassene.

### Forhold mellom klasser

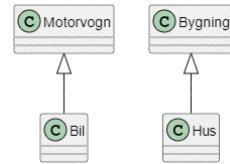
I UML er det fire forskjellige typer forhold mellom klasser.

Forhold	Forbindelse	PlantUML
Generalisering	◀	< --
Assosiasjon	◀	<--
• Aggregering	◊	o--
• Komposisjon	◆	*--
Avhengighet	↔	<..
Realisering	◀-----	< ..

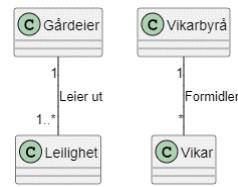
## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

**Arv** er en relasjon som kalles **generalisering** fordi attributter og metoder som er felles i subklasser, plasseres i en superklasse som subklassene arver. Forbindelsen angis som en linje med en tom trekant i enden. Dette kalles også en "ER-EN"-relasjon.. En bil **er en** motorvogn. Et hus **er en** bygning. I koden angir vi navnet på superklassen inni parenteser når vi definerer subklassen.

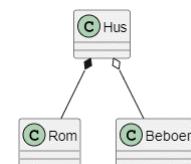


Når en klasse er **assosiert** med en annen klasse, oppstår det en forbindelse mellom dem. I koden kjennetegnes denne forbindelsen ved at objekter av den ene klassen har variabler som refererer til objekter av den andre klassen. Det gir oss tilgang til attributter og metoder i den andre klassen. Assosiasjonen angis med en strek og gis et navn. En gårdeier **Leier ut** en leilighet. Et vikarbyrå **Formidler** vikarer. Hvis forbindelsen går bare én vei, har streken en pil i enden. Vi kan også angi hvor mange objekter det er på hver side av forholdet. Assosiasjon er den vanligste formen for relasjoner mellom klasser og dermed viktigere enn arv med tanke på gjenbrukbar kode. I tillegg til vanlig assosiasjon finnes det to spesialtilfeller. De kalles aggregering og komposisjon.



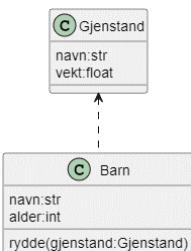
*	0 eller flere
1..*	1 eller flere
0..1	0 eller 1
4	Akkurat 4
4..8	4 til 8
4,6,8	Akkurat 4, 6 eller 8

**Komposisjon** kan vi tenke på som en "BESTÅR-AV"-relasjon, der en klasse består av en annen klasse. Et hus **består av** flere rom. Rommene er avhengig av huset for å eksistere. De kan opprettes inni Hus-klassen, og forsvinner når huset slutter å eksistere <sup>22</sup>. Tilsvarende som for en mappe i filutforskeren. Når vi fjerner mappen, forsvinner alle filene i den. Komposisjon og assosiasjon som en strek med et fylt rutersymbol i enden.



**Aggregering** kan vi tenke på som en "HAR-EN"-relasjon, der en klasse har én eller flere objekter av en annen klasse. Et hus **har en** eller flere beboere. Beboerne kan opprettes og bør ha referanser på utsiden av Hus-klassen så de ikke slettes når huset slutter å eksistere. Aggregering skiller seg ikke vesentlig fra en vanlig assosiasjon bortsett fra det viser hva som er hoved-objekt og hva som er del-objekt. Aggregering angis som en strek med et tomt rutersymbol i enden. Når vi bruker komposisjon eller aggregering i stedet for vanlig assosiasjon, er det ikke nødvendig å gi assosiasjonen et navn.

Vi sier at den første klassen er avhengig av en andre klassen dersom endringer i den andre klassen kan medføre at vi må gjøre endringer i den første klassen. **Avhengighet** kan kjennetegnes i kode ved at metoder i den første klassen benytter objekter fra den andre klassen som parametere. Barnet rydder bort gjenstander. Det kan godt hende vi må endre koden til barnets **rydde**-metode dersom vi fjerner **vekt**-attributtet fra **Gjenstand**-klassen. Avhengighet angis som en stiplet linje med en pil i enden. Det er viktig å velge hvilke relasjoner vi ønsker å vise i diagrammene, da det kan bli svært komplekst



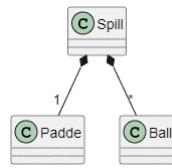
<sup>22</sup> Et objekt ligger lagret i datamaskinens minne så lenge det finnes variabler som peker på det. Når ingen variabler peker på objektet lenger, blir det ubrukelig for programmet (Hvorfor?) og slettes automatisk av en [garbage collector](#) for å frigi minneplass.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Hvis vi inkluderer alle. Dessuten er det ikke nødvendig å vise avhengighet hvis det allerede er en generalisering eller assosiasjon mellom klassene.

Hvis vi analyserer Multipong, kommer vi til at spillet bruker både baller og en padde. Dette er avhengighet. Vi kan også si at spillet **består av** en padde, flere baller, med mer. Padden og ballene er det behov for bare så lenge spillet eksisterer. Så spillet kan opprette ballene og padden, og det er greit at de blir borte når Spill-objektet slutter å eksistere. Dermed er det mer riktig å beskrive relasjonen som assosiasjon, nærmere bestemt komposisjon.

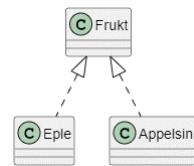


**Realisering** er mer uhåndgripelig og kjennetegnes i kode med **abstrakte klasser** og **grensesnitt**. Realisering kan ses på som en avtale. Vi bestemmer hva som skal gjøres på et sted, og hvordan det gjøres et annet sted. I abstrakte klasser og grensesnitt spesifiseres kun navnet på metodene med parametere, men uten en kodeblokk. Klassene som bruker disse abstrakte klassene eller grensesnittene må inneholde koden som mangler. Dette gir en løsere kobling og mindre avhengighet mellom klasser, noe som gjør systemene mer fleksible og enklere å vedlikeholde.

Det en del forskjeller mellom programmeringsspråkene som bruker statiske (Java, C#,...) og dynamiske typer (Python, Ruby,...):

- I Python bruker vi [duck typing](#) for polymorfisme (at objekter fra forskjellige klasser har felles metoder og kan behandles likt, selv om koden i metodene varierer), mens i Java bruker vi grensesnitt for å oppnå det samme, spesielt når klasser ikke kan dele en superklasse.
- Python har ikke grensesnitt, men vi kan oppnå noe lignende ved bruk av abstrakte klasser.
- I Python kan en subklasse arve fra flere superklasser selv om dette ikke nødvendigvis brukes til polyformisme. I Java kan en subklasse bare ha én superklasse, men det begrenser ikke polyformisme, for vi kan bruke grensesnitt i stedet.

Vi begrenser oss her til å fastslå at det ikke går å lage objekter av abstrakte klasser. La oss si vi har epler og appelsiner som arver felles attributter og metoder fra superklassen Frukt. Vi vil at objektene skal være enten Eple- eller Appelsin-objekter, men aldri Frukt-objekter, som verken er epler eller appelsiner. Da kan vi gjøre Frukt-klassen abstrakt. I Python må vi da importere [ABC](#) og [abstractmethod](#) fra [abc](#) (*Abstract Base Classes*), gjøre Frukt-klassen til en subklasse av [ABC](#) og dekorere en metode (f.eks. `__init__`) med `@abstractmethod`. I modellen angis realisering som en stiplet linje med en tom trekant i enden.



Å ha kunnskap om de ulike relasjonene mellom klasser er det første skrittet mot å utvikle fleksible systemer som er enkle å vedlikeholde og tilpasse til endringer. Dette innebærer også å forstå når, hvorfor og hvordan vi best designer modeller med de tilgjengelige mulighetene. Det finnes design prinsipper og mønstre som hjelper oss med det. Jo større system, desto større konsekvenser får et dårlig design, og jo viktigere blir det med et godt design.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Attributter

De fleste attributtene er primitive datatyper som `str`, `int`, `float` og `bool`. Informasjon som programmet trenger å huske, må vi ta vare på i attributter. Kravspesifikasjonen med brukstilfellene er et godt utgangspunkt for å finne attributtene vi må ha med i modellen. Siden vi ikke har jobbet med brukstilfeller ennå, kan vi bruke substantivmetoden så lenge. Substantiver som ikke blir til klasser, er mulige attributter.

Variabler som peker på objekter fra andre klasser, som fremmednøkler, vises som relasjonslinjer i UML diagrammer. Hvis vi ønsker å vise flere detaljer, kan vi samtidig vise dem som attributter i klassen.

### Operasjoner

Det som skal bli til metoder, heter operasjoner i UML. Operasjoner er abstrakte spesifikasjoner mens metoder er konkrete utførelser i form av kode. Fra brukstilfellene kan vi lage sekvensdiagram, og fra sekvensdiagram kan vi finne operasjoner. Siden vi ikke er der ennå, kan vi analysere verbene i kravspesifikasjonen for mulige operasjoner.

### Ansvarsoppgaver

Noe av det første vi må ta stilling til er hvor vi skal opprette objekter. Hvilke klasser skal få ansvaret for å opprette hvilke objekter? Klassers ansvarsoppgaver, roller og samarbeid med andre klasser er en del av [Responsibility-Driven-Design](#). Forpliktelsene blir oppfylt av metoder. Det er også utviklet [GRASP](#)-prinsipper som støtter denne designteknikken. [Creator](#)-prinsippet anbefaler oss å la klasse B opprette objekter av klasse A dersom en eller flere av betingelsene nedenfor er oppfylt.

- B-objekter inneholder eller består av A-objekter
- B-objekter registrerer A-objekter
- B-objekter har nær nytte av A-objekter
- B-objekter har initialiseringsinformasjonen for å opprette A-objekter  
(verdiene som sendes som argumenter til A-klassens konstruktør `__init__`)

I spillet [Multipong](#) ser vi at [App](#)-klassen består av [Padde](#)- og [Ball](#)-objekter og dessuten tilfredsstiller flere av de andre kriteriene. Så vi lot [App](#)-klassen opprette [Padde](#)- og [Ball](#)-objektene.

[Information Expert](#) er et annet design prinsipp som går ut på å legge operasjonen (definere metoden) i den klassen som har informasjonen som er nødvendig for å utføre oppgaven. Vi kan se dette prinsippet brukt i [Multipong](#)-spillet hvor paddens `update`-metode er plassert i [Padde](#)-klassen. Når `update()` kjøres, beregnes paddens nye posisjon ut fra nåværende posisjon og fart. Denne informasjonen ligger nettopp i [Padde](#)-klassen.

Et siste design prinsipp i denne runden er [Controller](#) som sier vi bør styre brukergrensesnittet fra programmets "rot"- eller "hoved"-objekt. I [Multipong](#)-spillet håndterer vi derfor tastetrykkene i [App](#)-klassen. Denne sentraliserte strukturen gjør koden lettere å lese, forstå, videreutvikle og feilsøke, noe som bidrar til et godt og ensartet brukergrensesnitt.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

Klassediagram er et av de mest brukte diagrammene av de 14 forskjellige UML-diagramtypene. Det inneholder navn på klassen, attributtene og metodene. Klasser forholder seg til hverandre gjennom ulike relasjoner: generalisering (arv), assosiasjon (inkludert mer spesifikke typer som komposisjon og aggregering), avhengighet og realisering. Relasjonene tegnes som streker mellom klassene, og symboler i enden av strekene viser hvilken type forbindelse det er. Substantivmetoden hjelper oss med å dele et problem opp i klasser.

*Responsibility-Driven Design* og GRASP (Creator, *Information Expert* og *Controller*) er designprinsipper for å tilordne ansvar, opprette objekter og styre brukergrensesnittet. Ved å følge disse prinsippene, er det stor sjanse for at systemene blir fleksible og enkle å vedlikeholde.

### Øvingsoppgaver

#### 3.5.1

Avgjør hvilken relasjon det er mellom klassene i programmene nedenfor:

a)

```
1 class Sjåfør():
2     def __init__(self, navn:str):
3         self.navn = navn
4         print('Ny sjåfør: ' + navn)
5
6 class Varebil():
7     def __init__(self, reg_nr:str):
8         self.reg_nr = reg_nr
9         self.sjåfør = None
10        print('\nNy varebil: ' + reg_nr)
11
12    def bruk_sjåfør(self, sjåfør:Sjåfør)->None:
13        self.sjåfør = sjåfør
14        print('\nNå er ' + self.sjåfør.navn + ' sjåfør'
15              '\npå varebil ' + self.reg_nr)
16
17 varebil = Varebil('AJ77309')
18 sjåfør = Sjåfør('Stig')
19 varebil.bruk_sjåfør(sjåfør)
```

```
Ny varebil: AJ77309
Ny sjåfør: Stig
```

```
Nå er Stig sjåfør
på varebil AJ77309
```

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

b)

```
1 class Robot():
2     def __init__(self, navn:str):
3         self.navn = navn
4         print('Ny robot: ' + navn)
5         self.armer = []
6         self.armer.append(Arm('høyre'))
7         self.armer.append(Arm('venstre'))
8
9 class Arm():
10    def __init__(self,navn:str):
11        self.navn = navn
12        print('Ny ' + self.navn + ' arm')
13
14 robot = Robot('R2-D2')
15 print([x for x in dir(robot) if x[0:2] != '__'])
✓ 0.6s
Ny robot: R2-D2
Ny høyre arm
Ny venstre arm
['armer', 'navn']
```

c)

```
1 class Gave:
2     def __init__(self,navn:str):
3         self.navn = navn
4         print('Ny gave: ' + navn)
5
6 class Julenisse:
7     def __init__(self,navn:str):
8         self.navn = navn
9         print('Ny julenisse: ' + navn)
10
11 def pakk(self,gave:Gave)->None:
12     print(self.navn + ' pakker en ' + gave.navn)
13
14 klaus = Julenisse('Klaus')
15 gave = Gave('Google Chromecast')
16 klaus.pakk(gave)
✓ 0.1s
Ny julenisse: Klaus
Ny gave: Google Chromecast
Klaus pakker en Google Chromecast
```

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

d)

```
1 class Person():
2     def __init__(self, navn:str):
3         self.navn = navn
4         print('Ny person: ' + navn)
5
6     def hils(self):
7         print('Jeg heter ' + self.navn +
8             ' og er ' + self.__class__.__name__.lower())
9
10 class Elev(Person):
11     def __init__(self,navn:str):
12         super().__init__(navn)
13
14 class Lærer(Person):
15     def __init__(self,navn:str):
16         super().__init__(navn)
17
18 personer = []
19 personer.append(Elev('Hilde'))
20 personer.append(Lærer('Knut'))
21 for person in personer: person.hils()
✓ 0.4s
Ny person: Hilde
Ny person: Knut
Jeg heter Hilde og er elev
Jeg heter Knut og er lærer
```

Vurderingsseksempler

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

e)

```
1 from abc import ABC, abstractmethod
2
3 class Konto(ABC):
4     @abstractmethod
5         def __init__(self,konto_nr:str):
6             self.konto_nr = konto_nr
7             print('\nNy ' + self.__class__.__name__ +
8                 ': ' + konto_nr)
9
10 class Sparekonto(Konto):
11     maks_antall_uttak = 12
12
13     def __init__(self, konto_nr:str):
14         super().__init__(konto_nr)
15         print('Maks antall uttak: ' +
16             str(Sparekonto.maks_antall_uttak))
17
18 class BSU_konto(Konto):
19     maks_årlig_sparebeløp = 27500
20
21     def __init__(self, konto_nr:str):
22         super().__init__(konto_nr)
23         print('Maks årlig sparebeløp: ' +
24             str(BSU_konto.maks_årlig_sparebeløp))
25
26 karis_konto = Sparekonto('1234 56 78999')
27 olas_konto = BSU_konto('4343 55 67677')
✓ 0.8s

Ny Sparekonto: 1234 56 78999
Maks antall uttak: 12

Ny BSU_konto: 4343 55 67677
Maks årlig sparebeløp: 27500
```

Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

f)

```
1 from random import shuffle
2
3 class Prosjekt():
4     def __init__(self, navn:str, medlemmer:list):
5         self.navn = navn
6         print('Nytt prosjekt: ' + navn)
7         self.medlemmer = medlemmer
8
9     def vis_prosjektgruppe(self)->None:
10        print('\nMedlemmer i prosjektet ' + self.navn + ' er')
11        navn = ', '.join([medlem.navn for medlem in self.medlemmer])
12        før, _, etter = navn.rpartition(', ')
13        print(før + ' og ' + etter)
14
15 class Ansatt():
16     def __init__(self,navn:str):
17         self.navn = navn
18         print('Ny ansatt: ' + navn)
19
20 ansatte = []
21 ansatte.append(Ansatt('Erik'))
22 ansatte.append(Ansatt('Frida'))
23 ansatte.append(Ansatt('Gustav'))
24 ansatte.append(Ansatt('Henriette'))
25 shuffle(ansatte)
26 prosjekt = Prosjekt('Julebord',ansatte[0:3])
27 prosjekt.vis_prosjektgruppe()
✓ 0.6s
Ny ansatt: Erik
Ny ansatt: Frida
Ny ansatt: Gustav
Ny ansatt: Henriette
Nytt prosjekt: Julebord

Medlemmer i prosjektet Julebord er
Frida, Erik og Henriette
```

#### 3.5.2

Lag UML klassediagrammer til programmene i oppgave 3.5.1.

#### 3.5.3

En idrettsklubb har 10 medlemmer - 5 jenter og 5 gutter. En kveld skal de arrangere en 4x100 meter stafettkonkurranse med to lag. Hvert av lagene vil bli trukket tilfeldig blant klubbens medlemmer. Utøverne skal løpe etappene i den rekkefølgen de blir trukket ut. De to medlemmene som ikke blir trukket ut, vil starte stafetten og ta tiden på lagene.

Jentene løper 100 meter på mellom 11.5 og 13.5 sekunder, mens guttene bruker mellom 11.0 og 13.0 sekunder.

Lag en modell og et program som simulerer stafetten.

- a) Bruk substantivmetoden til å bestemme klasser og attributter i modellen.

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

- b) Bestem forholdene mellom klassene.
- c) Hvilke operasjoner må med?
- d) Tegn et UML klassediagram for modellen.
- e) Lag et program som skriver ut etappetidene, total tid og hvilket lag som vant.
- f) (Tilleggsoppgave - valgfri) Lag et program med GUI og 8 figurer som står oppstilt og flytter seg som det var en stafett langs en rett linje.

Husk å bruke `random.uniform(a, b)` for å generere tilfeldige desimaltall mellom `a` og `b` for løpetidene til utøverne. Medlemslisten kan stokkes med `random.shuffle(x)`.

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 3.6 Dokumentasjon

#### Bolkens innhold

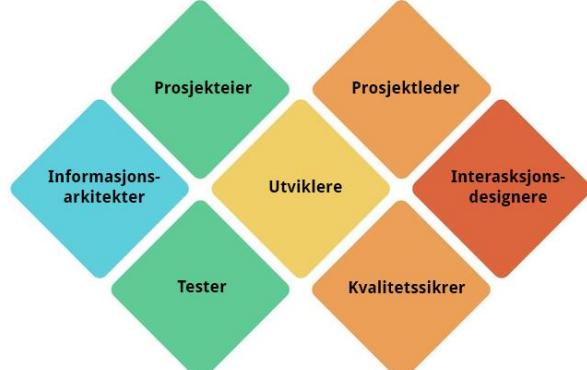
Innledning .....	192
Kommentarer .....	193
Typeantegnelser.....	194
dir, help, pydoc og IntelliSense .....	194
Docstring .....	195
Dokumentasjonsverktøy .....	196
Ta med minst dette.....	197
Øvingsoppgaver.....	198

#### Kompetanse mål:

- velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

#### Innledning

Dokumentasjon er dokumenter som kreves for noe. Det kan være et bevis for at noe er gjort, som en kvittering bekrefter at noe er betalt, eller det kan være et hjelpemiddel, som en brukerveiledning. Hvilke dokumenter og informasjon kan det bli behov for når vi utvikler datasystemer? Det kommer an på hvor stort prosjektet er, og kanskje hvem vi spør, for det er flere [roller](#) forbundet med prosjekter generelt, og



systemutviklingsprosjekter spesielt. De som skal bruke systemet, trenger annen informasjon enn de som skal utvikle det. Det samme gjelder gevinstansvarlig, prosjektleder, løsningsarkitekt, osv. Vi skrev i bok [1.8 Modellering](#) at systemutvikling omfatter flere faser. Det finnes ingen egen dokumentasjonsfase, for alle fasene dokumenteres. Vi har allerede jobbet med ting som inngår i dokumentasjonen. Klassediagram inngår når vi dokumenterer designfasen, resultatet av testene inngår når vi dokumenterer testfasen. I denne bolken skal vi jobbe med dokumentasjonen som inngår når vi koder i gjennomføringsfasen. Senere skal vi lære om kravspesifikasjon og brukstilfeller som inngår i analysefasen og brukerdokumentasjon i utrullingsfasen.

I [Manifesto for Agile Software Development](#) står det: "... we have come to value:... Working software over comprehensive documentation...". Det er dermed ikke sagt at vi dropper dokumentasjon når vi velger en smidig tilnærming, men at vi begrenser den til "så vidt god nok". Ikke for mye, men heller ikke for lite. Dokumentasjonen i tradisjonell systemutvikling

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

(*SDLC – System Development Life Cycle*) som fossefallmodellen kan fort bli unødvendig omfattende eller i uoverensstemmelse med koden (Hvorfor?).

Vi **kommenterer** kode for å gjøre den lettere å **vedlikeholde**, for oss selv og andre utviklere. Vi **dokumenterer** kode for å gjøre den lettere å **bruke**, for de vi samarbeider med og andre brukere. Både kommentarer i koden og dokumentasjon av koden er nyttig for de vi samarbeider med, og vi skal bruke verktøy som hjelper oss med begge deler. Denne bolken inngår derfor i kompetansemålet om å velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre, jfr. [KM09](#).

### Kommentarer

Det er skrevet mye om hvordan vi bør kommentere programmer, og en gjenganger er å skrive selvdokumenterende kode uten kommentarer: "En grei huskeregel er at vi skal skrive så tydelig kode at andre programmerere skal kunne forstå den uten å måtte trenge kommentarer. I noen tilfeller vil det likevel være hensiktsmessig å kommentere koden.". Se [Kommentering av kode, NDLA](#). Vi kan komme langt med bare å velge fornuftige navn på variabler og funksjoner, men hvis vi ikke skriver kommentarer, får vi ikke fortalt hvorfor vi gjør noe som vi gjør. Det finnes nok av råd om hva vi ikke skal gjøre, for eksempel:

- Ikke dupliser koden med unødvendige kommentarer.  
**DRY-principle, (Don' Repeat Yourself)**

[The Pragmatic Programmer](#), Dave Thomas og Andy Hunt

```
x = 23
x = 23 # Sett x til 23.
x = 23 # x er alder.
alder = 23
```

- Ikke bruk en kommentar når vi kan bruke en funksjon eller en variabel.

[Clean Code](#), Robert C. Martin.

```
if alder >= 18:
    pass # Personen er myndig.

er_myndig = alder >= 18
if er_myndig:
    pass
```

- Ikke forklar hvordan, men hvorfor.

[Code Complete](#), Steve McConnell

```
# Skjøt # med en hex-koden som blir stripsett for 0x
# og fylt på med nuller foran så den blir 6 karakterer.
farge = "#"+hex(randint(0, 2**24 - 1))[2:].zfill(6)

# Endre formatet på hex-koden så den kan brukes
# av fill parameteren i canvas.create_oval().
farge = "#"+hex(randint(0, 2**24 - 1))[2:].zfill(6)
```

I den prisbelønte boka [Code Complete](#) skriver Steve McConnell at vi kan innlede en kodeblokk med en kommentar som forteller hva vi ønsker å oppnå.

Komentarene skal ikke forklare eller gjenta det som står i koden, men klargjøre hvilken hensikt vi har. Samme sted i boka refererer McConnell til en undersøkelse utført av IBM. Den viste at de som vedlikeholdt programmer, oftest sa at det vanskeligste problemet var å forstå hensikten til den opprinnelige programmereren.

```
def vis_grafikk(self) -> None:
    self.canvas.delete("all")
    # Vis start- og målstreker.
    for i in range(5):
        x = X_START + i * ETAPPELENGDE_PX + 15
        self.canvas.create_line(x, 85, x, 215, fill="black")
    # Plasser lagene under hverandre
    for bane_nr, lag in enumerate(self.stafett.lag, start=1):
        y = BANE_AVSTAND * bane_nr
        # Plasser løperne bortover i hver sin x-posisjon
        for etappe_nr, etappe in enumerate(lag.etapper, start=1):
            self.canvas.create_image(
                etappe.x, y, image=self.figurer[bane_nr-1])
            # Vis navn og etappetider
            fullført_etatte = etappe.x > X_START + ETAPPELENGDE_PX * etappe_nr
            x = ETAPPELENGDE_PX * etappe_nr + 15
            if fullført_etatte:
                self.canvas.create_text(
                    x, y+35, text=f"{etappe.loper.navn}: " + f"{etappe.tid:.2f}")
            else:
                self.canvas.create_text(x, y+35, text=f"{etappe.loper.navn}")
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Videre bør vi være sparsomme med bruken av kommentarer på slutten av kodelinjer, for det blir fort rotete og distraherende, [PEP 8](#)<sup>23</sup> – Style Guide for Python Code.

For å få oss på rett spor, har vi tatt med noen flere sitater av kjente IKT-personer:

- "Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do." [Literate Programming](#), Donald E. Knuth
- "Don't comment bad code – rewrite it." [The Elements of Programming Style](#), Brian W. Kernighan og P. J. Plauger
- "Readability counts", [The Zen of Python](#) ([import this](#)), Tim Peters.

**Better-Comments**-utvidelsen i VS Code bruker tagger til å gi forskjellige typer kommentarer ulik formatering for å lettere skille dem fra hverandre. Taggene \*, !, ? og todo skrives etter #-tegnet. Vi kan legge til egne tagger og skreddersy formettingen. Når vi koder, er det vanlig å kommentere uferdig eller tvilsom kode eller raske løsninger med ord som TODO, HACK, FIXME og UNDONE ([Task Comments](#)). Disse kommentarene er for eget bruk og fjernes før vi sender koden fra oss.

```
# Vanlig kommentar
# * Fremhevet
# ! Advarsel
# ? Spørsmål
# TODO: Nøe som må gjøres
# FIXME: Nøe virker ikke som det skal og må fikses
# HACK: En rask og ufiks løsning på et problem
# UNDONE: Nøe som har blitt fjernet og hvorfor
```

### Typeantegnelser

Vi vet fra før at Python bruker dynamiske typer så en variabel får typen til objektet den peker på. Peker variabelen på 4, så får den typen `int`. Tilordner vi den til `True`, får den typen `bool`, osv. Vi kan sende en hvilken som helst type som argument i et funksjonskall, for parameteren får typen til objektet som argumentet peker på. Hvis funksjonen forventer en `str`, men får en `float`, kan det oppstå problemer når funksjonen kjøres (Hvorfor?). For å gjøre det lettere å bruke riktig type til et argument, kan vi angi typen etter parameternavnet og et kolon (:) i funksjonsdefinisjonen. Typen blir ikke kontrollert, men den vises i IntelliSense-vinduet som dukker opp når vi fører musepekeren over funksjonsnavnet. Dermed får vi en påminnelse om hvilken type argumentet må være. Typeantegnelsene kommer også med i dokumentasjonen når vi bruker verktøy til å lage [Docstring-er](#). Se nest siste avsnitt i denne bolken.

### dir, help, pydoc og IntelliSense

Start python fortolkeren ved å skrive `python` i terminalvinduet. `dir()` returner en liste med tilgjengelige moduler, klasser og objekter, blant annet `__builtins__`, modulen som inneholder de innebygde funksjonene (BIF:– [Built-in Functions](#)).

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>>
```

De får vi listet ut med `dir(__builtins__)`. I den listen vil vi se blant annet `len`. Dokumentasjonen for `len` vises ved å skrive `help(len)`.

```
>>> help(len)
Help on built-in function len in module builtins:
len(obj, /)
    Return the number of items in a container.
```

<sup>23</sup> PEP står for *Python Enhancement Proposal*

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi skriver `help("modules")`, får vi se alle tilgjengelige moduler, blant annet `math`. Så kan vi skrive `import math` og `dir(math)` for å se innholdet i `math`, blant annet `pow`. Dokumentasjonen for `pow` vises ved å skrive `help(pow)`, og vi kan sjekke at `math.pow(2,10)` som er det samme som `2**10`.

```
>>> import math
>>> help(math.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> math.pow(2,10)
1024.0
>>> 2**10
1024
```

`help()` bruker `pydoc` som frembringer dokumentasjon fra python-moduler. `pydoc`, som også er en modul, kan vi kjøre som et skript direkte fra terminalvinduet: `python -m pydoc math.pow`. Resultatet blir det samme. Vi kan til og med starte en lokal webserver og en nettleser som viser dokumentasjonen på websider:

`python -m pydoc -b.`



Når vi fører musepekeren over `pow` i VS Code viser IntelliSense informasjonen som ser ganske lik ut. Så hvor kommer all denne dokumentasjonen fra?

```
(function) pow(__x: _SupportsFloatOrIndex, __y: _SupportsFloatOrIndex, /) -> float
import math
math.pow
Return x**y (x to the power of y).
```

### Docstring

`help`, `pydoc` og `IntelliSense` henter dokumentasjonen fra *docstring*-er som ligger lagret i koden. En *docstring* er tekst som starter og slutter med tre anførselstegn (""""")<sup>24</sup>.

```
1 import math
2 print(math.pow.__doc__)
3
4 0.4s
Return x**y (x to the power of y).
```

*Docstring*-en må være den første setningen i en modul, funksjons, klasse eller metode. Spesialattributtet `__doc__` til objektet blir tilordnet innholdet i *docstring*-en. Derfor viser `print(math.pow.__doc__)` nesten det samme som `help(math.pow)`.

[PEP257 – Docstring Conventions](#) inneholder retningslinjer for *docstring*-er. Vi skiller mellom de som er på én linje og de som er på flere linjer. De på én linje, skal ikke ha blanke linjer før eller etter. De skal være på formen "Gjør dette", "Returner dette". *Docstring*-er som går over flere linjer skal ha en blank linje etter den første linjen som skal være et sammendrag. Generelt gjelder det at

- moduler skal ha med én linjes sammendrag for hver klasse, unntak og funksjon.
- klasser skal ha med et sammendrag og nevne offentlige metoder og objektvariabler.
- funksjoner skal ha med et sammendrag, argumenter, returverdier og unntak.

<sup>24</sup> Tre anførselstegn brukes også for tekst som går over flere linjer.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

La oss dokumentere funksjonen til høyre som returnerer partall mellom to verdier.

```
1 def partall(a:int, b:int)->list:  
2     return [x for x in range(a,b+1) if x%2==0]  
3  
4 print(partall(1,10))  
✓ 0.3s  
[2, 4, 6, 8, 10]
```

For å gjøre jobben lettere, bruker vi VS Code-utvidelsen **autoDocstring** som frembringer mye av teksten for oss. Etter vi har skrevet tre anførselstegn, kommer det fram en snarveismeny hvor vi kan velge *Generate Docstring*. Det finnes flere *docstring*-formater, og vi bruker [Google Docstring Format](#), som er et av de vanligste formatene.

```
def partall(a: int, b: int) -> list:  
    """Lag partall. Returner dem i en liste  
  
    Examples:  
        >>> partall(0,5)  
        [0, 2, 4]  
        >>> partall(1,10)  
        [2, 4, 6, 8, 10]  
  
    Args:  
        a (int): Nedre grense (fra og med)  
        b (int): Øvre grense (til og med)  
  
    Returns:  
        list: partall fra a til b  
    """  
    return [x for x in range(a, b+1) if x % 2 == 0]
```

```
Help on function partall in module __main__:  
  
partall(a: int, b: int) -> list  
    Lag partall. Returner dem i en liste  
  
    Examples:  
        >>> partall(0,5)  
        [0, 2, 4]  
        >>> partall(1,10)  
        [2, 4, 6, 8, 10]  
  
    Args:  
        a (int): Nedre grense (fra og med)  
        b (int): Øvre grense (til og med)  
  
    Returns:  
        list: partall fra a til b
```

I følge [PEP 8](#) standarden dokumenterer vi offentlige metoder. Metoder som ikke er offentlige, holder det med å kommentere.

Det kan også være lurt å få hjelp til å formtere koden. Høyre-klikk i dokumentet, velg **Format Document With...** fra snarveismenyen og deretter **Python**.

Format Document With...  
Format Selection Ctrl+K Ctrl+F

Ordet *linting*, som betyr å fjerne lo fra tekstiler, brukes i programering for å oppdage syntaksfeil og brudd på kodestandarder. Som vi så i [1.7 Feil](#), kunne Pylance finne syntaksfeil. Hvis vi ønsker å analysere koden for å sikre at den overholder kodestandarder, må vi bruke en *linter*, som **Pylint**-utvidelsen. Hvis vi har installert Pylint, vil vi få advarsler dersom vi ikke skriver *docstrings*.

### Dokumentasjonsverktøy

Bruk **pydoc** med **-w** for å lagre en webside av *docstring*-en. Websiden kan vi skrive ut i PDF-format hvis vi ønsker det.

```
progs ➤ python -m pydoc -w .\b_3_6_partall.py  
wrote b_3_6_partall.html
```

**b\_3\_6\_partall**

**Functions**

<b>partall</b> (a: int, b: int) -> list Lag partall. Returner dem i en liste  Examples: >>> partall(0,5) 6.0 >>> partall(1,8) 6.0  Args: a (int): Nedre grense (fra og med) b (int): Øvre grense (til og med)  Returns: list: partall fra a til b
---

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Det er mer vanlig å bruke andre verktøy som [pdoc](#) (enkel) eller [Sphinx](#) (avansert).

Websidene som lages av *docstring*-ene er nettopp den API-dokumentasjonen vi trenger. [pdoc](#) kan, som [pydoc](#), vise dokumentasjonen på websider

```
pip install pdoc  
pdoc --docformat google ./b_3_6_2_partall.py
```

Når vi er fornøyd, kan vi eksportere dokumentasjonen til HTML filer med

```
pdoc --docformat google ./b_3_6_2_partall.py -o ./docs
```

I tillegg til API-dokumentasjonen er det behov for flere dokumenter. Det kan være lisensbetingelser, installasjonsveiledning, brukermanual, med mer. [pdoc](#) og [Sphinx](#) lar oss sy alt i sammen, men vi også bruke verktøy som [Jekyll](#).



API Documentation  
for Python Projects



Jupyter, NumPy, pandas, seaborn, og matplotlib bruker alle Sphinx.

Jupyter: Try Jupyter | Usage | Projects | Community | Contributing | More ▾  
NumPy: User Guide | API reference | Development | Release notes | Learn ↗  
pandas: Getting started | User Guide | API reference | Development | Release notes  
seaborn: Installing | Gallery | Tutorial | API | Releases | Citing | FAQ  
matplotlib: Plot types | Examples | Tutorials | Reference | User guide | Develop | Release notes

Til slutt må vi gjøre dokumentasjonen tilgjengelige for andre. Vi kan ha den på egne websider eller bruke tjenster som [GitHub Pages](#) eller [Read the Docs](#). Kombinasjonen av Jekyll og GitHub er populær fordi dokumentasjonen oppdateres automatisk når koden endres.



Denne bolken har stort sett dreid seg om dokumentasjon i forbindelse med koding (gjennomføringsfasen). Vi minner om at dette er bare en del av dokumentasjon som er nødvendig i et IKT-prosjekt, men viktig er den:

*If It Isn't Documented, It Doesn't Exist. ([Coding Horror](#))*

#### **Ta med minst dette**

Kommentarer forenkler vedlikehold av koden for oss selv og andre. God praksis tilskir at vi skal skrive kode så tydelig at andre utviklere kan forstå den uten behov for omfattende kommentarer. Når vi kommenterer, bør vi unngå å gjenta det koden allerede sier, heller forklare hvorfor vi gjør noe og tydeliggjøre hensikten bak koden.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Dokumentasjon gjør programmet enklere å bruke for andre. **pdoc** er et program som automatisk bruker *docstrings* til å lage dokumentasjon. *Docstrings* er tekstblokker mellom tre anførselstegn i koden, som for eksempel `'''Returnerer kvadratroten av et gitt tall.'''`, og de forklarer både hva koden gjør og hvordan den skal brukes, enten det gjelder en hel applikasjon, enkelte klasser, metoder eller funksjoner. pdoc omdanner disse tekstuksjonene til websider som tjener som dokumentasjon framfor trykte manualer.

#### Øvingsoppgaver

##### 3.6.1

- Kommenter og lag *docstring*-er i koden til oppgave 3.5.3.e **eller** 3.5.3.f.  
Hvis du ikke har løst de oppgavene, kan du bruke løsningsforslaget eller et annet program du har laget.
- Lag API-dokumentasjon med **pdoc** til oppgave 3.6.1 a). Legg ved mappen `./docs` i besvarelsen.

##### 3.6.2

- Endre `b_3_6_partall.py` slik at funksjonen ikke feiler dersom brukeren anvender feil type i argumentene. Returner en standard liste med partall fra 2 til og med 10 dersom det oppstår et unntak.
- Dokumenter endringen i *docstring*-en.

Vurderingsseksemplar

## 3.7 Enhetstesting av objektorienterte programmer

### Bolkens innhold

Innledning .....	199
Noen definisjoner.....	200
Introduksjon til objektorientert enhetstesting .....	201
Testing av opprettelse og initialisering av objekter .....	201
Testing av metoder som returnerer verdier .....	201
Testing av metoder med bivirkninger .....	202
Testing av metoder som håndterer unntak .....	203
Bruk av <i>fixtures</i> for testkontekst.....	205
Testsamlinger og kjøring av utvalgte tester.....	206
Refaktorering og forbedring av kode etter testing .....	207
Automatisert testdokumentasjon med pytest-html.....	207
Ta med minst dette.....	208
Øvingsoppgaver.....	209

### Kompetanse mål:

- vurdere og bruke strategier for feilsøking og testing av programkode
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

### Innledning

I bok 1.7 om feil, lærte vi å bruke [assert](#)-setninger til å sjekke at koden vår fungerer som den skal. Hvis den ikke gjør det, lager programmet et unntak og avbrytes. Hvis vi bruker *debuggeren*, stopper utførelsen der hvor betingelsen feiler, og vi kan sjekke innholdet i variablene. I bok 2.5 [Programvaretesting](#) ble vi kjent med forskjellige typer tester som blir utført på ulike nivåer, testdrevet utvikling, og vi lagde våre første, automatiske enhetstester.

I denne bolken skal vi lære mer om enhetstester, spesielt for objektorienterte programmer. Vi skal fortsette å bruke [pytest](#), som går for å være mer populært enn [unittest](#). [pytest](#) har mange fordeler framfor [unittest](#), men er særegent for Python. [unittest](#) følger med Python og er ganske likt ([xUnit](#)-) testverktøy i andre programmeringsspråk. Det kan gjøre en overgang fra Python til et annet språk enklere om man kjenner til [unittest](#). De som skal lære seg Python og er kjent med xUnit, kan likevel bruke [pytest](#) som også [støtter](#) [unittest](#). Når vi bruker [pytest](#) i stedet for [unittest](#), bytter vi ut det mer generelle med det enkle. I [5.11 Unittest](#) (Tilleggsstoff) viser vi hvordan den objektorienterte testen i avsnittet Testkontekst kan utføres med [unittest](#).

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Noen definisjoner

**Testenhet** (*test unit*): Enhetstester skrives og kjøres av den som koder, for å forsikre seg om at deler av programmet fungerer som planlagt. En del eller testenhet kan være en hel modul, men er som oftest en funksjon, klasse eller metode.

**Testtilfelle** (*test case*): Et generelt begrep som beskriver hva som testes og hvordan det testes. Et testtilfelle består av inndata, hvordan testen utføres og forventet resultat. Det kan implementeres som en testfunksjon eller en testmetode innenfor en testklasse.

**Testfunksjon** (*test function*): En frittstående funksjon som representerer et enkelt testtilfelle. Testfunksjoner brukes for å teste individuelle funksjoner i en modul og plasseres i egne testmoduler. Start funksjonsnavnet med `test_` for at den skal bli gjenkjent pytest.

**Testmetode** (*test method*): En metode innenfor en testklasse som representerer et enkelt testtilfelle. Start metodenavnet med `test_` for at den skal bli gjenkjent av pytest. En testmetode tester vanligvis én spesifikk metode i en klasse, men vi kan ha flere testmetoder for samme metode, for å teste ulike egenskaper.

**Testklasse** (*test class*): En klasse som inneholder en eller flere testmetoder. Hver testmetode er et testtilfelle, og testklassen grupperer disse sammen. Dette er nyttig for å organisere tester som også kan dele samme testkontekst. Start klassenavnet med `Test` for at den skal bli gjenkjent av pytest.

**Testmodul** (*test module*): En Python-skriptfil som inneholder testklasser og/eller testfunksjoner. En testmodul samler tester som hører sammen. Start filnavnet med `test_` (`test_*.py`) for at den skal bli gjenkjent av pytest.

**Testsamling** (*test suite*): En samling av flere testtilfeller som kjøres samtidig. Testsamlinger effektiviserer testprosessen og gjør det mulig å kjøre mange tester sammen eller spesielle utvalg av tester for å sjekke spesifikke egenskaper ved programvaren.

**Testkontekst** (*test context*): Miljøet eller tilstanden som må settes opp før testene kan kjøres og oppryddingen etter at testene er utført. Dette kan være å klargjøre nødvendige data, innstillinger eller ressurser som testene er avhengige av. I pytest håndteres dette ved hjelp av `fixtures`.

**Fixtures:** Funksjoner som kjører før og/eller etter testene for å sette opp og rydde opp testkonteksten. De brukes for å unngå duplisering av kode som setter opp testmiljøet. I pytest angir vi dette med dekoratøren `@pytest.fixture` foran definisjonen av funksjonen.

**Etterligning** (*Mocking*): Når vi tester en metode, kan den noen ganger være avhengig av annen kode som vi ikke kan kontrollere, som funksjoner som returnerer tilfeldige verdier, løpende aksjekurser, eller nettverkstilgang. Etterligning betyr å erstatte slike deler av koden med imitasjoner (*mocks*) som oppfører seg som de originale komponentene. Dette gir kontrollerte og forutsigbare resultater. Se [4.8 Mer om testing](#) og [5.6 Etterligning \(mocking\) med pytest \(Tilleggsstoff\)](#) for mer informasjon.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

**Testdrevet utvikling** (*TDD – Test Driven Development*): innebærer å skrive testene først (som feiler), deretter skrive koden (slik at testene passerer), og til slutt omstrukturere koden (slik at den blir bedre). Se [Testdrevet utvikling](#) i 2.5 Programvaretesting

**Refaktorering:** Når alle testene passerer, går vi i gang med å forbedre koden. Vi prøver å gjøre den enklere, lettere å lese og vedlikeholde, uten å endre hvordan den fungerer.

### Introduksjon til objektorientert enhetstesting

Objektorientert enhetstesting handler om å teste individuelle klasser og metoder i objektorienterte programmer. Vi vil forsikre oss om at objektene oppfører seg som forventet, både når det gjelder tilstandene (attributtene) og handlingene (metodene). Testing av objektorienterte programmer skiller seg ikke vesentlig fra testing av prosedyreorienterte programmer, fordi metoder er som funksjoner, bare inni klasser. Vi kan teste objektorientert kode med et prosedyreorientert testprogram, men anbefaler å bruke testklasser for bedre organisering og lesbarhet.

Eksempelet viser en `bil.py`-modul med en `Bil`-klasse med attributtet `motor_status` og metodene `start_motor()` og `stopp_motor()`. Testprogrammet ligger i en testmodul `test_bil.py`. Det er organisert i en testklasse `TestBil` med testmetodene `test_start_motor()` og `test_stopp_motor()`.

The screenshot shows two code editors side-by-side. The left editor contains `bil.py` with a class `Bil` having methods `__init__`, `start_motor`, and `stopp_motor` which set the `motor_status` attribute to 'stoppet', 'startet', and 'stoppet' respectively. The right editor contains `test_bil.py` with a `TestBil` class containing two test methods: `test_start_motor` and `test_stopp_motor`. Each test creates a `Bil` object, calls the respective method, and asserts that `motor_status` is correctly updated. To the right of the editors is a terminal window showing the test results: 2/2 tests passed in 1.4s. The terminal also shows the file structure: `b_3_7_1.bil`, `test_bil.py`, `TestBil`, `test_start_motor`, and `test_stopp_motor`.

### Testing av opprettelse og initialisering av objekter

Vi fortsetter med en `bil.py`-modul med en `Bil`-klasse med attributtet `motor_status` og metodene `start_motor()` og `stopp_motor()`. Testprogrammet ligger i en testmodul `test_bil.py`, men nå med testmetoden `test_init()` i testklassen `TestBil`. Vi vil teste at objektene blir opprettet riktig; at de har blitt opprettet (`is not None`), er av riktig klasse (`isinstance`), og at attributtene har fått riktige verdier. `isinstance()` er en innebygd funksjon i Python som brukes til å sjekke om et objekt er en instans av en spesifikk klasse eller en subklasse av denne klassen. Her sjekker vi om `bil`-objektet er en instans av `Bil`-klassen (`assert isinstance(bil, Bil)`).

The screenshot shows two code editors side-by-side. The left editor contains `bil.py` with the same `Bil` class as before. The right editor contains `test_bil.py` with a `TestBil` class containing a `test_init` method. This method creates a `Bil` object and performs three assertions: 1) `bil` is not `None`; 2) `bil` is an instance of `Bil` (using `isinstance(bil, Bil)`); and 3) the `motor_status` attribute of `bil` is 'stoppet'. To the right of the editors is a terminal window showing the test results: 1/1 test passed in 1.0s. The terminal also shows the file structure: `b_3_7_2.bil`, `test_bil.py`, `TestBil`, and `test_init`.

### Testing av metoder som returnerer verdier

Når vi tester metoder i en klasse som returnerer verdier, er det viktig å sikre at disse verdiene er korrekte. Eksempelet viser en `elbil.py`-modul med en `Elbil`-klasse med attributtene `batteri_nivaa` og `forbruk_per_mil`, og metoden `bereggn_rekkevidde()`. Testprogrammet ligger i en

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

testmodul `test_elbil.py`. Det er organisert i en testklasse `TestElbil` med testmetodene `test_beregn_rekkevidde()`, `test_beregn_rekkevidde_flatt_batteri()`, `test_beregn_rekkevidde_effektiv()` og `test_beregn_rekkevidde_ineffektiv()`.

The screenshot shows two code files: `# elbil.py` and `# test_elbil.py`. The `elbil.py` file defines a class `Elbil` with an `__init__` method that initializes `batteri_nivaa` and `forbruk_per_mil`. It also has a `beregn_rekkevidde` method that returns the range as a float. The `test_elbil.py` file contains four test cases for `TestElbil`: `test_beregn_rekkevidde` (asserts range for 50 kWh at 2.0 km per kWh), `test_beregn_rekkevidde_flatt_batteri` (asserts range for 0 kWh at 2.0 km per kWh), `test_beregn_rekkevidde_effektiv` (asserts range for 60 kWh at 1.5 km per kWh using `pytest.approx`), and `test_beregn_rekkevidde_ineffektiv` (asserts range for 50 kWh at 2.5 km per kWh). To the right, a UML class diagram for `Elbil` is shown with attributes `batteri_nivaa` and `forbruk_per_mil`, and methods `__init__`, `beregn_rekkevidde`, `kjør(mil)`, `skriv_til_fil(filnavn)`, and `skriv_til_skjerm`. Below the code and diagram is a screenshot of a test runner showing 4/4 tests passed.

- Separate testmetoder gir bedre feilsøking, lesbarhet og isolasjon av tester, noe som gjør dem enklere å vedlikeholde og mer pålitelige enn én testmetode med flere `assert`-setninger.
- Bruk av `pytest.approx` brukes for å sammenligne flyttall som ikke er eksakt like. Vi kan velge relativt eller absolutt avvik. Standard er relativt avvik med `rel=1e-6` (en millionandel). Her har vi valgt `abs=0.1`, som vil si at verdien kan avvike med  $\pm 0.1$ , altså mellom 33.2 og 33.4.

### Testing av metoder med bivirkninger

I enhetstesting kan vi ofte stå overfor metoder som ikke returnerer noen verdi. Slike metoder har vanligvis bivirkninger, eller "noe som skjer på utsiden av metoden som følge av metodekallet", noe som gir dem en hensikt. Klassediagrammet til høyre viser `Elbil`-klassen med dens attributter og metoder. Merk at `void` i UML tilsvarer `None` i Python, noe som indikerer at metodene ikke returnerer noen verdi. Her tester vi tre eksempler:

- endringer i en attributtverdi (`batteri_nivaa`)
- skriving til en fil
- vanlig utskrift til terminal med `print`



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Når vi tester funksjoner som skriver til filer, er det viktig at disse testene ikke endrer eller legger igjen filer på datamaskinen vår. Da kan vi bruke [tmp\\_path](#) som er en ferdiglagd *fixture*. Den gir oss en midlertidig mappe hvor vi trygt kan opprette og endre filer under testingen, uten at det påvirker resten av systemet. Som standard lagres filene fra de tre siste testkjøringene i midlertidige mapper lokalisert i

`Brukere\<brukernavn>\AppData\Local\Temp` i Windows og i `/var/folders` i macOS.

```
# elbil.py
from pathlib import Path

class Elbil:
    def __init__(self, batteri_nivaa, forbruk_per_mil):
        self.batteri_nivaa = batteri_nivaa      # i kWh
        self.forbruk_per_mil = forbruk_per_mil   # i kWh per mil

    def kjør(self, mil):
        self.batteri_nivaa -= self.forbruk_per_mil * mil
        if self.batteri_nivaa < 0:
            self.batteri_nivaa = 0   # Batteriet kan ikke gå under 0.

    def skriv_til_fil(self, filnavn):
        sti = Path(__file__).parent.resolve()
        filnavn = sti.joinpath(filnavn)
        with open(filnavn, 'w', encoding='utf-8') as fil:
            fil.write(f'Batterinivå: {self.batteri_nivaa} kWh\n')

    def skriv_til_skjerm(self):
        print(f'Batterinivå: {self.batteri_nivaa} kWh')
```

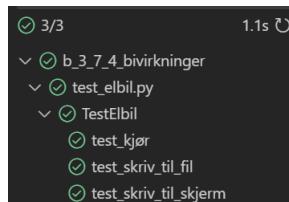
```
# test_elbil.py
from elbil import Elbil

class TestElbil:
    def test_kjør(self):
        elbil = Elbil(50, 2.0)
        elbil.kjør(10)
        # 50 kWh - (2.0 kWh per mil * 10 mil) = 30 kWh
        assert elbil.batteri_nivaa == 30

    def test_skriv_til_fil(self, tmp_path):
        elbil = Elbil(50, 2.0)
        filnavn = tmp_path.joinpath('test_batterinivaa.txt')
        elbil.skriv_til_fil(filnavn)
        with open(filnavn, 'r', encoding='utf-8') as fil:
            innhold = fil.read()
        assert innhold == 'Batterinivå: 50 kWh\n'

    def test_skriv_til_skjerm(self, capsys):
        elbil = Elbil(50, 2.0)
        elbil.skriv_til_skjerm()
        captured = capsys.readouterr()
        assert captured.out == 'Batterinivå: 50 kWh\n'
```

`print` produserer utdata som går via operativsystemets standard output (`stdout`) og standard error (`stderr`) datastrømmer. Disse rutes normalt til terminalvinduet og kan fanges opp av `Capys` i `pytest`, slik at vi får testet innholdet. `capys` er en `pytest fixture` som må angis som argument. `capsys.readouterr()` returnerer et objekt som inneholder alt som ble skrevet ut til både `stdout` og `stderr`. Dette objektet har to egenskaper: `out` for det som



ble skrevet til `stdout`, og `err` for det som ble skrevet til `stderr`. Her kunne vi brukt bare `capsys.readouterr().out`.

### Testing av metoder som håndterer unntak

Feilhåndtering i et program vil si å håndtere situasjoner når ting går galt på en kontrollert måte, slik at programmet ikke krasjer, og brukeren får nyttig tilbakemelding. Unntak kastes av operativsystemet eller av oss ved hjelp av `raise`, for å indikere at en feil har oppstått. Vi kaster egne unntak i metodene for å gi spesifikke feilmeldinger og forhindre uventet oppførsel. Unntak fanges av oss med `try` og `except` i koden som kaller disse metodene, for å forhindre at programmet krasjer og for å gi brukeren tilbakemelding om hva som gikk galt. Programmet kan da be brukeren om ny input, slik at de kan rette opp feilen og prøve igjen.

Når en `assert`-setning i en test feiler, lages et `AssertionError`-unntak, og testen feiler. Testen feiler også dersom metoden vi tester feiler, men hva gjør vi når metoden er ment å feile? Når vi skal teste om unntakshåndteringen vi selv har skrevet, fungerer riktig, kaller vi metodene med argumenter som forårsaker unntakene. Når programmet vi tester kaster unntak, vil testen feile med et annet unntak enn `AssertionError` hvis vi ikke sier at vi forventer et unntak. Det gjør vi med `pytest.raises`. `pytest.raises` brukes i en *context manager* (`with`-blokk) for å fange opp unntaket og verifisere at det kastes som forventet.

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Til å demonstrere unntakshåndtering har vi laget en ny variant av kjør-metoden til elbilen. Før vi oppdaterer gjenværende rekkevidde etter kjøringen, sjekker vi tre ting:

1. Argumentet mil må være oppgitt som et tall (`int` eller `float`).
2. Det kan ikke være negativt.
3. Det kan ikke være større enn gjenværende rekkevidde.

I alle disse tilfellene kaster programmet et unntak med en nyttig tilbakemelding.

```
# elbil.py
class Elbil:
    def __init__(self, batteri_nivaa, forbruk_per_mil):
        self.batteri_nivaa = batteri_nivaa # i kWh
        self.forbruk_per_mil = forbruk_per_mil # i kWh per mil

    def beregn_rekkevidde(self):
        return self.batteri_nivaa / self.forbruk_per_mil

    def kjør(self, mil):
        if not isinstance(mil, (int, float)):
            raise TypeError("Antall mil må være en int eller float")
        if mil < 0:
            raise ValueError("Antall mil kan ikke være negativt")
        rekkevidde = self.beregn_rekkevidde()
        if mil > rekkevidde:
            raise ValueError("Ikke nok strøm på batteriet til å kjøre så langt")
        forbruk = self.forbruk_per_mil * mil
        self.batteri_nivaa -= forbruk

    def koble_til_lader(self):
        ikke_implementert = True
        if ikke_implementert:
            raise NotImplementedError("Lader ikke implementert")
        return "Koblet til lader"
```

```
# test_elbil.py
import pytest
from elbil import Elbil

class TestElbil:
    def test_kjør_negativ_mil(self):
        elbil = Elbil(100, 2.0)
        # Metode 1: Bruker match-parameteren
        with pytest.raises((ValueError, TypeError), match="Antall mil kan ikke være negativ"):
            elbil.kjør(-5)

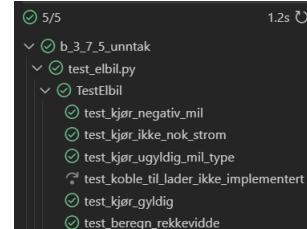
    def test_kjør_ikke_nok_strom(self):
        elbil = Elbil(100, 2.0)
        # Metode 2: Bruker e.value for å sjekke feilmeldingen
        with pytest.raises(ValueError) as e:
            elbil.kjør(60)
        assert "Ikke nok strøm på batteriet til å kjøre så langt" in str(e.value)

    def test_kjør_ugyldig_mil_type(self):
        elbil = Elbil(100, 2.0)
        with pytest.raises(TypeError, match="Antall mil må være en int eller float"):
            elbil.kjør("ti")

    @pytest.mark.xfail(reason="Ikke implementert")
    def test_koble_til_lader_ikke_implementert(self):
        elbil = Elbil(batteri_nivaa=100, forbruk_per_mil=2)
        assert elbil.koble_til_lader() == "Koblet til lader"

    def test_kjør_gyldig(self):
        elbil = Elbil(100, 2.0)
        elbil.kjør(10)
        assert elbil.batteri_nivaa == 80 # 100 - (2.0 * 10) = 80 kWh

    def test_beregn_rekkevidde(self):
        elbil = Elbil(100, 2.0)
        assert elbil.beregn_rekkevidde() == 50 # 100 kWh / 2.0 kWh per mil = 50 mil
```



```
test_elbil.py::TestElbil::test_kjør_negativ_mil PASSED
test_elbil.py::TestElbil::test_kjør_ikke_nok_strom PASSED
test_elbil.py::TestElbil::test_kjør_ugyldig_mil_type PASSED
test_elbil.py::TestElbil::test_koble_til_lader_ikke_implementert XFAIL (Ikke implementert)
test_elbil.py::TestElbil::test_kjør_gyldig PASSED
test_elbil.py::TestElbil::test_beregn_rekkevidde PASSED
```

===== 5 passed, 1 xfailed in 0.13s =====

Når vi tester unntakene med `pytest.raises()`, må vi spesifisere hvilket unntak vi forventer. Det er mulig å bruke en tuppel for å angi flere unntakstyper. Vi kan også bruke `match`-parameteren for å sjekke meldingen som følger med unntaket. En alternativ metode er å ta vare på unntaksobjektet med `as e`, noe som gir oss større fleksibilitet til å teste flere egenskaper ved unntaket. I eksempelet ovenfor tester vi kun teksten som følger med unntaket ved å bruke en `assert`-setning med `str( e.value)`.

Vurderingssempolar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Biblioteksrutiner kaster også unntak hvis vi bruker dem feil. I `math`-modulen finner vi `sin`- og `asin`-funksjonene. Vi kan finne sinus til hvilken som helst vinkel, fra minus til pluss uendelig. Verdimengden ligger imidlertid mellom -1 og 1, så hvis vi går den andre veien med `asin`, må argumentet også ligge innenfor dette intervallet. `asin(2)` er utenfor definisjonsmengden og feiler (`ValueError: math domain error`), som beskrevet i [C18](#)-standarden.

Vi så i [2.5 Programvaretesting](#) at vi kan markere testfunksjoner som vi forventer vil feile med `@pytest.mark.xfail`. Et vanlig eksempel er en test for en funksjon som ennå ikke er implementert, eller en kjent feil som ennå ikke er rettet. Her tester vi metoden `koble_til_lader`, som ikke er implementert. Testen vil da ikke feile, men markeres med **XFAIL** (*eXpected to FAIL*), og testsamlingen som helhet vil fortsatt passere. Når en test passerer til tross for at den er merket som forventet å feile, markeres den med **XPASS**.

### Bruk av *fixtures* for testkontekst

En testkontekst er miljøet hvor vi gjennomfører testene. Det er alt som må være på plass før vi kan utføre testene. Dette kan være oppretting av data, midlertidige mapper, innstillinger eller nødvendige ressurser som nettverksforbindelser, databaser eller servere. Etterpå må vi rydde opp for å sikre at senere tester ikke påvirkes. Vi kan spare tid og unngå å skrive kode om igjen ved å gjenbruke testkonteksten.

I Pythons unittest og lignende [xUnit](#)-systemer bruker vi metoder som `setUp` og `tearDown` for å ta oss av oppsett og opprydding. `setUp` utføres før hver test for å klargjøre, og `tearDown` tar seg av opprydding etterpå. I [5.7 Unittest](#) (Tilleggsstoff) viser vi hvordan dette gjøres med kode.

I pytest bruker vi derimot [\*fixtures\*](#) for å håndtere testkonteksten mer fleksibelt. Disse spesielle funksjonene, markert med `@pytest.fixture`, aktiveres ved å inkludere funksjonsnavnet som argument i testene. Ved hjelp av `scope`-parameteren bestemmer vi når oppsett og opprydding utføres – enten for hver funksjon, klasse, modul, pakke, eller for hele testøkten (`function` som er standard, `class`, `module`, `package` eller `session`). Det som kommer foran `return` i en `fixture` fungerer som `setUp`. Skal vi ha med `tearDown`, må vi bruke `yield` i stedet, og alt som kommer etter `yield` er opprydding.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
# test_bil.py
import pytest
from bil import Bil

@pytest.fixture(scope="class")
def cls():
    print("\nClass setup")
    yield
    print("\nClass teardown")

@pytest.fixture
def bil(cls):
    print("\nFunction setup")
    bil = Bil()
    yield bil
    print("\nFunction teardown")

class TestBil:
    def test_start_motor(self, bil):
        print("Tester start av motor")
        bil.start_motor()
        assert bil.motor_status == 'startet'

    def test_stopp_motor(self, bil):
        print("Tester stopp av motor")
        bil.start_motor()
        bil.stopp_motor()
        assert bil.motor_status == 'stoppet'

# bil.py
class Bil:
    def __init__(self):
        self.motor_status = 'stoppet'

    def start_motor(self):
        self.motor_status = 'startet'

    def stopp_motor(self):
        self.motor_status = 'stoppet'

# Smidig IT-2 © TIP AS 2024
```

```
test_bil.py::TestBil::test_start_motor
Class setup
Function setup
Tester start av motor
PASSED
Function teardown

test_bil.py::TestBil::test_stopp_motor
Function setup
Tester stopp av motor
PASSED
Function teardown

Class teardown

=====
2 passed in 0.07s =====
```

Ovenfor viser vi en enkel bruk av *fixtures* for eksempelet i introduksjonsavsnittet. *Fixture*en for testklassen gjør ingenting, men er kun med for å demonstrere bruk av *fixtures* på klassenivå. `print`-setningene er kun tatt med for å vise når de blir kjørt. Kjør `pytest -s -v` fra terminalvinduet for å se utskriften.

### Testsamlinger og kjøring av utvalgte tester

Vi kan samle flere testtilfeller i en testsamling som dekker det vi ønsker å teste. Noen ganger ønsker vi kanskje å kjøre bare et utvalg av testtilfellene, og andre ganger alle sammen. I utgangspunktet ønsker vi at testene skal dekke 100% av koden, men det kan være en økonomisk grense for hvor omfattende testene skal være. Pytest lar oss velge ut tester og testsamlinger på ulike nivåer

- En enkel funksjon (`pytest bane/test_modul.py::test_function`)
- All testene i en modul (`pytest bane/test_modul.py`)
- En enkel metode (`pytest bane/test_modul.py::TestClass::test_method`)
- Alle testene i en klasse (`pytest bane/test_modul.py::TestClass`)
- Alle testene i en mappe (`pytest bane`)
- Tester som velges ut etter mønster i filnavnene (`pytest -k ...`)
- Tester som er kategorisert med markeringer (`@pytest.mark....`)

Når vi skriver bare `pytest` i terminalvinduet, vil `pytest` automatisk oppdage og kjøre alle testene i den nåværende katalogen og alle dens undermapper. For å kjøre bare `test_stopp_motor()` i foregående eksempel, kunne vi ha skrevet `pytest -s -v ./test_bil.py::TestBil::test_stopp_motor`.

I VS Code er det dessuten lett å velge hvilke tester som skal kjøres fra det grafiske brukergrensesnittet.

```
test_bil.py::TestBil::test_stopp_motor
Class setup
Function setup
Tester stopp av motor
PASSED
Function teardown
Class teardown

=====
1 passed in 0.04s =====
```

# Smidig IT-2

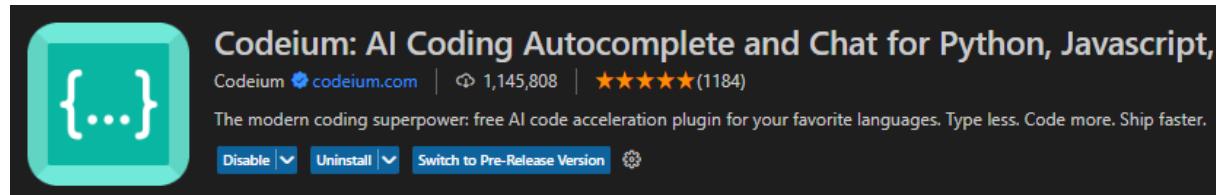
## for eksklusiv bruk ved <navn på skole>

### Refaktorering og forbedring av kode etter testing

Når alle testene passerer, går vi i gang med å forbedre koden. Først retter vi opp alle feilene som er avdekket gjennom testingen. Dette sikrer at koden fungerer som forventet før vi begynner på refaktorering.

Refaktorering vil si å gjøre koden enklere og lettere å lese og vedlikeholde, uten å endre på hva den gjør. Vi prøver å fjerne duplisert kode, forbedre navngivning og kommentarer, og omstrukturere koden for bedre lesbarhet og modularitet. Dette er en viktig del av [testdrevet utvikling \(TDD\)](#), som vi diskuterte i bolken [2.5 Programvaretesting](#).

Det finnes også verktøy som kan hjelpe oss med dette. [GitHub Copilot](#) og [Sourcery](#) koster penger, men vi kan bruke [Codeium](#) som er gratis. Alle kommer som utvidelser til VS Code



I **bil.py (b\_3\_7\_7\_refaktorering)** skriver vi status til en loggfil når motoren startes eller stoppes. Merk koden og velg *Codeium: Refactor Selected Code Block* fra *Command Palette*... (Ctrl+Shift+P) og deretter *Make this faster and more efficient*. Vi ser at Codeium foreslår å trekke ut den felles kodelogikken for å skrive til loggfilen i en egen metode, noe som gjør koden mer effektiv og lettere å vedlikeholde. Samtidig foreslår Codeium en annen måte å aksessere filen på, ved å bruke `with_name` i stedet for `joinpath`.

The image shows two code snippets side-by-side. On the left is the original code (bil.py), and on the right is the refactored code (bil\_refaktorert.py). Both snippets define a class Bil with methods start\_motor and stopp\_motor that write to a log file. The refactored version uses `with_name` instead of `joinpath` and separates the logging logic into a `write_log` method. To the right of the code is a terminal window showing the contents of a file named bil.log, which contains four entries: "Motor startet", "Motor stoppet", "Motor startet", and "Motor stoppet".

```
# bil.py
from pathlib import Path

class Bil:
    def __init__(self, loggfil):
        sti = Path(__file__).parent.resolve()
        fil = sti.joinpath(loggfil)
        self.loggfil = fil
        self.motor_status = 'stoppet'

    def start_motor(self):
        self.motor_status = 'startet'
        with open(self.loggfil, 'a') as f:
            f.write(f"Motor startet\n")

    def stopp_motor(self):
        self.motor_status = 'stoppet'
        with open(self.loggfil, 'a') as f:
            f.write(f"Motor stoppet\n")

# bil_refaktorert.py
from pathlib import Path

class Bil:
    def __init__(self, loggfil):
        self.loggfil = Path(__file__).with_name(loggfil)
        self.motor_status = 'stoppet'

    def write_log(self, status):
        with open(self.loggfil, 'a') as f:
            f.write(f"Motor {status}\n")

    def start_motor(self):
        self.motor_status = 'startet'
        self.write_log('startet')

    def stopp_motor(self):
        self.motor_status = 'stoppet'
        self.write_log('stoppet')

bil.log
1 Motor startet
2 Motor stoppet
3 Motor startet
4 Motor stoppet
```

### Automatisert testdokumentasjon med pytest-html

`pytest-html` lager en HTML-rapport med testresultatene.

- Installer med `pip install pytest-html`.
- Lag rapporten med `pytest --html=testreport.html`

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Se i [dokumentasjonen](#) for hva som kan endres på. Vi har endret tittelen og skrevet et sammendrag. Koden for dette ligger i `conftest.py`. Husk å legge ved `assets`-mappen som inneholder `style.css`. Det er ikke nødvendig dersom vi bruker `--self-contained-html`.

Filene ligger mappen `b_3_7_8_testdokumentasjon`.

```
import pytest
@pytest.hookimpl(tryfirst=True)
def pytest_html_report_title(report):
    report.title = "My testreport"

def pytest_html_results_summary(
    prefix, summary, postfix):
    prefix.extend(["<p>My summary</p>"])
```

### My testreport

Report generated on 24-Jun-2024 at 00:48:36 by `pytest-html` v4.1.1

#### Environment

Python	3.11.4
Platform	Windows-10-10.0.22631-SP0
Packages	<ul style="list-style-type: none"><li>• pytest: 7.4.0</li><li>• pluggy: 1.2.0</li></ul>
Plugins	<ul style="list-style-type: none"><li>• anyio: 4.0.0</li><li>• html: 4.1.1</li><li>• metadata: 3.0.0</li><li>• mock: 3.12.0</li></ul>

#### Summary

My summary

2 tests took 3 ms.

(Un)check the boxes to filter the results.

Result ▲		Test	Duration	Links
Passed		test_bil.py::TestBil::test_start_motor	2 ms	
Passed		test_bil.py::TestBil::test_stopp_motor	2 ms	

### Ta med minst dette

Vi skrev vår første tester for pytest i bolken [2.5 Programvareutvikling](#). Disse var prosedyreorienterte. To funksjoner i en programmodul og to testfunksjoner i en testmodul. Nå har vi testet objektorienterte programmer med metoder i en klasse i en programmodul, som vi har testet med testmetoder i en testklasse i en testmodul. Vi har gått grundigere til verks og testet opprettelsen av objekter (`isinstance()`), returverdier fra metoder, oppdatering av attributter, utskrift til terminalvinduet, utskrift til fil og unntakshåndtering. Vi har brukt `fixtures` for å få på plass det som er nødvendig før vi kjører testene, og for å rydde opp etterpå (etter `yield`). Testsamlinger lar oss velge ut hvilke tester vi vil kjøre samtidig. Når vi har rettet opp i alle feilene, fortsetter vi å forbedre koden ved å refaktorere. Til dette brukte vi også AI-verktøyet Codeium. Til slutt så vi hvordan vi kunne lage testdokumentasjon med `pytest-html`.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## Øvingsoppgaver

### 3.7.1

- a) Hva er hensikten med enhetstesting?
- b) Hva er en testenhet?
- c) Hvordan kan vi organisere testtilfeller i pytest?
- d) Hvilke ulike nivåer kan vi velge å kjøre tester på ved hjelp av pytest?
- e) Hva er en *fixture* i pytest, og hvordan brukes den?
- f) Hvorfor kan det være nyttig å etterligne deler av koden under testing?
- g) Hvilke funksjoner i unittest-superklassen brukes for forberedelser og opprydding i testtilfellene?
- h) Hvordan kan vi lage en HTML-rapport med testresultater ved hjelp av pytest-html?

### 3.7.2

I denne oppgaven skal vi lage en klasse med metoder for å analysere tekst og deretter teste denne klassen ved hjelp av pytest.

1. Opprett mappen `tekstanalyse`
2. I `tekstanalyse`-mappen, opprett
  - a. To undermapper: `src` og `tests`, se [pytest Good Integration Practices](#)
  - b. Filen `pytest.ini` med innholdet:

```
[pytest]
pythonpath = src
```

for at pytest skal finne programmet
3. I `src`-mappen, opprett filen `tekst_info.py` med klassen `TekstInfo` hvor følgende metoder skal implementeres:
  - a. `finn_antall_karakterer(tekst:str)->int:`
  - b. `finn_antall_ord(tekst:str)->int:`
  - c. `finn_lengste_ord(tekst:str)->str:`
  - d. `finn_korteste_ord(tekst:str)->str:`

Metodene tar inn en tekststreng som argument og returner henholdsvis antall karakterer, antall ord, lengste og korteste ord i teksten.
4. I `tests`-mappen, opprett filen `test_tekst_info.py` og implementer en test for hver metode ved å bruke pytest.

```
py test_tekst_info.py > ...
import pytest
from tekst_info import TekstInfo
```
5. Ta skjermbilde etter å ha kjørt testene både med GUI i VS Code og CLI i terminalvinduet. Vis disse skjermbildene som en del av besvarelsen din.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

6. Bruk [pytest-html](#) til å generere en HTML-testrapport med dokumentasjon for testresultatene

3.7.3

#### Oppgave 6 [IT-2 Eksamensoppgaver](#)

Vi ønsker å lage en liten kalkulator for de fire regneoperasjonene – minus, pluss, gange og dele. Nedenfor finner du pseudokode som beskriver en del av en løsning ved hjelp av fire funksjoner, vist til høyre.

- a) Lag et klassediagram for en tilsvarende klasse kalt Kalkulator til bruk i en objektorientert løsning.
- b) Implementer klassen i ditt programmeringsspråk.
- c) Implementer et egnet testprogram for å teste løsningen, og identifiser mulige feil og unntak.
- d) Implementer nødvendig håndtering av mulige feil og unntak.

```
FUNCTION pluss(a, b)
    RETURN a + b
ENDFUNCTION

FUNCTION minus(a, b)
    RETURN a - b
ENDFUNCTION

FUNCTION gange(a, b)
    RETURN a * b
ENDFUNCTION

FUNCTION dele(a, b)
    RETURN a / b
ENDFUNCTION
```

## 3.8 Fortsettelse av Pandas

### Bolkens innhold

Innledning .....	211
Repetisjon.....	211
Manglende data.....	212
Komplekse beregninger med apply .....	214
Omforme tabeller med melt og pivot .....	215
Kombiner tabeller med concat og merge.....	218
iterrows.....	220
Ta med minst dette.....	220
Øvingsoppgaver.....	220

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

### Innledning

I bok [3.1 Jupyter, NumPy og pandas](#) lærte vi grunnleggende bruk av **pandas**-biblioteket. La oss begynne med en kort repetisjon for å friske opp hukommelsen, før vi går dypere inn i mer avanserte emner som manglende data, omforming med **melt/pivot**, kombinering med **concat/merge**, og iterering med **iterrows**.

### Repetisjon

Filen [b\\_3\\_8\\_1\\_repetisjon.ipynb](#) inneholder de viktigste begrepene og teknikkene som vi har jobbet med tidligere, inkludert

- Lese inn data fra csv- og json-filer
- Informasjon om datasettet (datatyper, antall non-null, enkel statistikk)
- Velge ut bestemte kolonner, rader og celler
- Legge til og fjerne kolonner og rader
- Sortering, gruppering og funksjoner

Husk at Pandas bruker **Series** for endimensjonale tabeller og **DataFrame** for todimensjonale tabeller. Disse datastrukturene inneholder mange flere metoder enn innebygde objekter som **list** og **dict**, og derfor må de ofte behandles annerledes. Pandas skiller seg også fra **NumPy** ved å gi mer omfattende funksjonalitet som er spesielt egnet til å håndtere strukturerede data, inkludert håndtering av tekst og manglende data, og funksjoner for å gruppere, filtrere og aggregere data. Når det gjelder gruppering, har vi mange aggregasjonsfunksjoner å velge

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

mellan. Disse inkluderer innebygde funksjoner som kan brukes direkte på grupper, for eksempel `count()`, `sum()`, `mean()`, `median()`, `min()`, `max()`, `first()` og `last()`. Vi kan finne mer informasjon om disse funksjonene i Pandas API-dokumentasjonen under [Built-in aggregation methods](#).

### Manglende data

Når det gjelder håndtering av manglende data, er det viktig å vurdere ulike strategier som fjerning av observasjoner eller utfylling av verdier med funksjonen `fillna()`. Manglende data er et vanlig problem i datavitenskap og kan oppstå av ulike årsaker, for eksempel tekniske feil, utelatte svar eller bevisst ikke-innsamling av data. Manglende verdier kan representeres som `Nan` (Not a Number) eller `NA` (not available) i datasettet. Håndtering av manglende data er viktig for å unngå skjevheter og feil i analysen. Det finnes ulike strategier for å håndtere dette problemet.

Først kan vi bruke `isna()` for å identifisere manglende data. For eksempel gir `df.isna().sum()` antall manglende verdier i hver kolonne. Vi kan også finne totalt antall manglende celler, prosentandelen av manglende celler, samt rader og kolonner med minst én manglende verdi. Dette gir en god oversikt over omfanget av manglende data i datasettet.

En vanlig strategi er å fjerne observasjoner med manglende verdier ved hjelp av funksjonen `dropna()`, men dette kan føre til redusert antall observasjoner og dermed redusert nøyaktighet i analysen. En annen strategi er å fylle inn manglende verdier med en bestemt verdi, ved hjelp av funksjonen `fillna()`. Dette kan imidlertid føre til en forvrengning av resultatene, spesielt hvis det er en stor andel manglende verdier i datasettet. En annen metode er å bruke `interpolate()` for å estimere manglende verdier basert på omkringliggende data.

Mange forskjellige strategier kan brukes for å håndtere manglende data. Det er viktig å være bevisst på hvilken strategi som passer best for det aktuelle datasettet og analysen som skal utføres. En grundig analyse av datakvaliteten og hva som kan ha forårsaket manglende verdier kan hjelpe til med å bestemme den mest hensiktsmessige strategien for å håndtere manglende data.

Her viser vi et eksempel på hvordan det kan gjøres og hvordan det påvirker utregningene, se filen `b_3_8_2_manglende_data.ipynb`.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Manglende data

Les inn csv filen med pandas

```
import pandas as pd
df = pd.read_csv('b_3_8_2_data.csv',sep=';')
display(df.head())
print(df.info())
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0
1	Bjørn	NaN	180.0	80.0
2	Christian	35.0	NaN	70.0
3	Dina	42.0	175.0	NaN

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   column   Non-Null Count  Dtype  
--- 
 0   Navn      4 non-null     object  
 1   Alder     3 non-null    float64 
 2   Høyde    3 non-null    float64 
 3   Vekt      3 non-null    float64 
dtypes: float64(3), object(1)
memory usage: 260.0+ bytes
None
```

```
Oversikt over manglende data
```

```
print(f"Antall manglende verdier i hver kolonne:\n{df.isna().sum()}")
total_missing = df.isna().sum().sum()
print(f"\nTotalt antall manglende verdier i df: {total_missing}")
total_cells = df.size
missing_percentage = (total_missing / total_cells) * 100
print(f"Prosentandel av manglende verdier i df: {missing_percentage:.2f}%")
rows_with_na = df.isna().any(axis=1).sum()
print(f"Antall rader i df med minst én manglende verdi: {rows_with_na}")
columns_with_na = df.isna().any(axis=0).sum()
print(f"Antall kolonner i df med minst én manglende verdi: {columns_with_na}")
```

```
Antall manglende verdier i hver kolonne:
Navn      0
Alder     1
Høyde    1
Vekt      1
dtype: int64

Totalt antall manglende verdier i df: 3
Prosentandel av manglende verdier i df: 18.75%
Antall rader i df med minst én manglende verdi: 3
Antall kolonner i df med minst én manglende verdi: 3
```

Vi mangler **Alder** for **Bjørn**, **Høyde** for **Christian** og **Vekt** for **Dina**.

Fjern aller rader med na og regn ut gjennomsnittshøyden

```
1 df_dropna_alle = df.dropna()
2 display(df_dropna_alle.head())
3 gjsn = df_dropna_alle ['Høyde'].mean()
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0

Gjennomsnittshøyde: 165.0 cm

Fjern rader som mangler høyde og regn ut gjennomsnittshøyden

```
1 df_dropna_høyde = df.dropna(subset=['Høyde'])
2 display(df_dropna_høyde.head())
3 gjsn = df_dropna_høyde ['Høyde'].mean()
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0
1	Bjørn	NaN	180.0	80.0
3	Dina	42.0	175.0	NaN

Gjennomsnittshøyde: 173.3 cm

Hvis vi fjerner alle radene som mangler data, sitter vi igjen med **Anne** og gjennomsnittshøyden blir **165 cm**. Hvis vi fjerner bare de radene som mangler høydeverdier, sitter vi igjen med **Anne, Bjørn** og **Dina** og gjennomsnittshøyden blir **173.3 cm**.

Gjennomsnitt og standardavvik av orginaltabellen.

```
1 display(df)
2 gjsn = df['Høyde'].mean()
3 std = df['Høyde'].std()
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
5 print(f'Std. avvik: {std:.1f} cm')
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0
1	Bjørn	NaN	180.0	80.0
2	Christian	35.0	NaN	70.0
3	Dina	42.0	175.0	NaN

Gjennomsnittshøyde: 173.3 cm  
Std. avvik: 7.6 cm

Gjennomsnitt og standardavvik av df\_dropna

```
1 gjsn = df_dropna_alle['Høyde'].mean()
2 std = df_dropna_alle['Høyde'].std()
3 display(df_dropna_alle)
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
5 print(f'Std. avvik: {std:.1f} cm')
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0

Gjennomsnittshøyde: 165.0 cm  
Std. avvik: nan cm

Hvis vi ikke gjør noe med tabellen, regner pandas ut gjennomsnitt og standardavvik fra de radene som ikke mangler data. Hvis vi fjerner alle radene med **NaN**, står vi igjen med én verdi, 165 cm, og standardavvik er meningsløst.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Gjennomsnitt og standardavvik av df\_dropna\_høyde

```
1 display(df_dropna_høyde)
2 gjsn = df_dropna_høyde['Høyde'].mean()
3 std = df_dropna_høyde['Høyde'].std()
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
5 print(f'Std. avvik: {std:.1f} cm')
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.0	58.0
1	Bjørn	NaN	180.0	80.0
3	Dina	42.0	175.0	NaN

```
Gjennomsnittshøyde: 173.3 cm
Std. avvik: 7.6 cm
```

Hvis vi fjerner raden som mangler høyde, får vi samme resultat som for originaltabellen, men vi har mistet **Alder** og **Vekt** til Christian.

Fyll inn manglende høydeverdier med gjennomsnittet

```
1 df_fillna = df.copy()
2 df_fillna['Høyde'] = df_fillna['Høyde'].fillna(gjsn)
3 display(df_fillna.head())
✓ 0.0s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.000000	58.0
1	Bjørn	NaN	180.000000	80.0
2	Christian	35.0	173.333333	70.0
3	Dina	42.0	175.000000	NaN

Gjennomsnitt og standardavvik for df\_fillna

```
1 display(df_fillna)
2 gjsn = df_fillna['Høyde'].mean()
3 std = df_fillna['Høyde'].std()
4 print(f'Gjennomsnittshøyde: {gjsn:.1f} cm')
5 print(f'Std. avvik: {std:.1f} cm')
✓ 0.1s
```

	Navn	Alder	Høyde	Vekt
0	Anne	25.0	165.000000	58.0
1	Bjørn	NaN	180.000000	80.0
2	Christian	35.0	173.333333	70.0
3	Dina	42.0	175.000000	NaN

```
Gjennomsnittshøyde: 173.3 cm
Std. avvik: 6.2 cm
```

Hvis vi fyller ut de manglende høydeverdiene med gjennomsnittet av høydeverdiene, forblir gjennomsnittet det samme, men standardavviket går ned, noe som vil påvirke statistiske analyser.

Vi bør bruke **dropna** eller **fillna** når vi jobber med manglende data i pandas for å unngå feil og skjevheter i analysen. Selv om pandas kan håndtere manglende data, er det viktig å være bevisst på hvordan vi håndterer dette problemet for å sikre nøyaktige resultater.

### Komplekse beregninger med apply

I et tidligere eksempel så vi hvordan enkel regning kunne brukes til å doble verdier i en kolonne ved å skrive `df['E'] = df['A']*2`. Dette opprettet en ny kolonne **E** hvor hver verdi var det dobbelte av den tilsvarende verdien i kolonne **A**. Nå skal vi bruke **apply()** til å løse mer avanserte oppgaver, som å håndtere måleenheter. For eksempel, hvis vi har lengder oppgitt i forskjellige enheter, som meter (**m**), desimeter (**dm**), og centimeter (**cm**), må vi først konvertere disse til en felles enhet før vi kan summere dem.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
import pandas as pd
data = {'Lengde': ['10 cm', '2 m', '15 dm']}
df = pd.DataFrame(data)

def gjør_om_til_meter(value):
    # Fjern mellomrom og skille mellom tall og enhet
    verdi, enhet = value.split()
    # Konverter numerisk del til flyttall
    verdi = float(verdi)
    # Konverter basert på måleenhet
    if enhet == 'm':
        return verdi
    elif enhet == 'dm':
        return verdi / 10
    elif enhet == 'cm':
        return verdi / 100
    # Hvis enhet ikke er støttet, returner None
    else:
        return None

# Bruk funksjonen med apply
df['Lengde_m'] = df['Lengde'].apply(gjør_om_til_meter)
print(df)
print(f'Summen av lengdene er {df["Lengde_m"].sum()} meter.')
```

	Lengde	Lengde_m
0	10 cm	0.1
1	2 m	2.0
2	15 dm	1.5

Summen av lengdene er 3.6 meter.

Eksempelet viser at `apply()` kan brukes til å utføre tilpassede operasjoner på data basert på spesifikke kriterier eller betingelser knyttet til verdier en `DataFrame`.

### Omforme tabeller med melt og pivot

Å omforme tabeller med `melt` og `pivot` kan være nyttig for å endre datalayout og gjøre det lettere å analysere og visualisere dataene. La oss begynne med følgende datasett for å demonstrerer omforming av tabeller, se filen [b\\_3\\_8\\_3\\_melt\\_og\\_pivot.ipynb](#).

Tabellen inneholder informasjon om salg av tre ulike produktkategorier (**Løpesko**, **Fotballsko** og **Tursko**) fordelt på tre år (**2020**, **2021** og **2022**) og tre ulike geografiske regioner (**Nord**, **Sør** og **Øst**).

```
1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 df = pd.DataFrame({'År': [2020, 2020, 2020, 2021, 2021, 2021, 2022, 2022, 2022],
6 'Region': ['Nord', 'Sør', 'Øst', 'Nord', 'Sør', 'Øst', 'Nord', 'Sør', 'Øst'],
7 'Løpesko': [70, 85, 110, 80, 100, 130, 90, 100, 120],
8 'Fotballsko': [90, 120, 180, 100, 140, 200, 110, 150, 220],
9 'Tursko': [140, 170, 220, 160, 200, 280, 180, 210, 260]})
10 display(df.head())
✓ 0.1s
```

	År	Region	Løpesko	Fotballsko	Tursko
0	2020	Nord	70	90	140
1	2020	Sør	85	120	170
2	2020	Øst	110	180	220
3	2021	Nord	80	100	160
4	2021	Sør	100	140	200

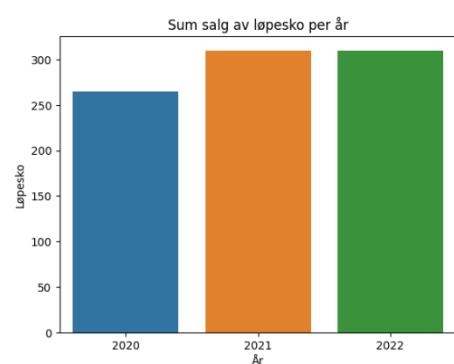
Tabellen inneholder informasjon

om salg av tre ulike produktkategorier (**Løpesko**, **Fotballsko** og **Tursko**) fordelt på tre år (**2020**, **2021** og **2022**) og tre ulike geografiske regioner (**Nord**, **Sør** og **Øst**).

```
1 sns.barplot(data=df, x='År', y='Løpesko',
2               estimator='sum', errorbar=None)
3 plt.title("Sum salg av løpesko per år")
4 plt.show()
```

Vi kan lage enkle diagrammer med

Seaborn uten å måtte omforme tabellen. Til høyre viser vi et søylediagram som viser summen av salget av **Løpesko** per år (for alle regionene samlet)



Husk at `estimator`-parameteren brukes til å aggregere dataene for oss. Hvis vi ikke spesifiserer noen estimator, vil Seaborn bruke gjennomsnittsverdien (`mean`) som standard estimator. Vi bruker `errorbar=None` for å fjerne de svarte strekene (konfidensintervallene) fra søylene.

## Smidig IT-2

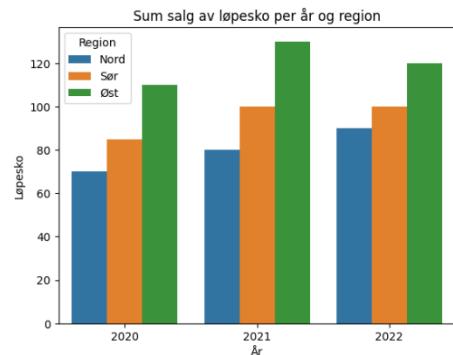
### for eksklusiv bruk ved <navn på skole>

```

1 sns.barplot(data=df, x='År', y='Løpesko', hue='Region',
2               estimator='sum', errorbar=None)
3 plt.title("Sum salg av løpesko per år og region")
4 plt.show()

```

Vi kan også lage enkelte grupperte søylediagram uten å måtte omforme tabellen. Søylediagrammet til høyre viser summen av salget av **Løpesko** per **År** og **Region**. Vi legger til **hue**-parametren for å gruppere etter **Region**.



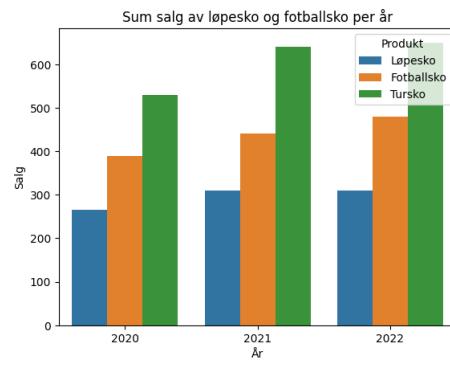
Verre blir det hvis vi ønsker å se salgstallene for alle produktene i en og samme figur. Da kan vi bruke **melt**-funksjonen og omforme en "bred" tabell (med flere kolonner som representerer forskjellige variabler) til en "lang" tabell (med én kolonne som representerer variabelnavn og en annen kolonne som representerer variabelverdier). Her beholder vi **År** og **Region** (**id\_vars**) og slår sammen de andre kolonnene, som også kunne vært spesifisert med **value\_vars=['Løpesko', 'Fotballsko', 'Tursko']**. **var\_name** blir navnet til den sammenslåtte kolonnen og **value\_name** navnet på kolonnen som inneholder verdiene. Nå kan vi vise salget for hvert år og gruppere etter produkt.

```

1 melt_df = pd.melt(df, id_vars=['År', 'Region'],
2                     var_name='Produkt', value_name='Salg')
3 display(melt_df.iloc[[0, 1, 9, 10, 18, 19]])

```

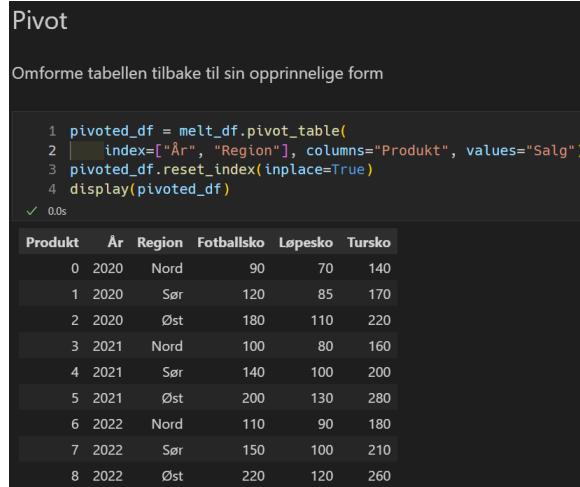
År	Region	Produkt	Salg	
0	2020	Nord	Løpesko	70
1	2020	Sør	Løpesko	85
9	2020	Nord	Fotballsko	90
10	2020	Sør	Fotballsko	120
18	2020	Nord	Tursko	140
19	2020	Sør	Tursko	170



Et alternativ til **melt** er å bruke **groupby**, men det ville vært mer tungvint.

Det går også an å gå den andre veien fra "langt" format til "breddt" format med **pivot**-funksjonen. Det kan være gunstig når vi har en stor mengde data og ønsker å få et bedre overblikk over hva dataene inneholder.

La oss først se at det går an å omforme tabellen tilbake til sin opprinnelige form med **pivot\_table**. **index** spesifiserer hvilke kolonner som skal brukes som indeks for den nye pivot-tabellen. **columns** spesifiserer hvilken kolonne (eller kolonner) som skal brukes som kolonner i den nye tabellen, og **value** spesifiser hvilken kolonne verdiene skal hentes fra. Vi kaller **reset\_index** etter **pivot\_table** for å flytte indeksen (som i dette tilfellet er en MultiIndex) tilbake til kolonner.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Hvis vi ønsker å få en bedre oversikt over den opprinnelige tabellen, kan vi bruke **pivot**. Vi ser at denne tabellen er mye lettere å lese enn den opprinnelige tabellen. Vi kan også formtere tabellen med CSS.

Vis en mer oversiktig tabell

```
1 df_pivot = df.pivot(index='År', columns='Region')
2 display(df_pivot)
```

✓ 0s

Region	Løpesko			Fotballsko			Tursko		
	Nord	Sør	Øst	Nord	Sør	Øst	Nord	Sør	Øst
<b>År</b>									
2020	70	85	110	90	120	180	140	170	220
2021	80	100	130	100	140	200	160	200	280
2022	90	100	120	110	150	220	180	210	260

Region	Løpesko			Fotballsko			Tursko		
	Nord	Sør	Øst	Nord	Sør	Øst	Nord	Sør	Øst
<b>År</b>									
2020	70	85	110	90	120	180	140	170	220
2021	80	100	130	100	140	200	160	200	280
2022	90	100	120	110	150	220	180	210	260

Vi kan også bruke **pivot** til å aggregere data for oss.

```
1 styles = {
2     'header': 'background-color: #003366; \
3                 color: white; font-weight: bold;',
4     'even': 'background-color: #f2f2f2; color: black;',
5     'odd': 'background-color: #d3d3d3; color: black;'
6 }
7
8 styled_df = df_pivot.style \
9     .set_properties(**{'text-align': 'center'}) \
10    .set_table_styles([
11        {'selector': 'th',
12         'props': styles['header']}
13    ], [
14        {'selector': 'tr:nth-child(even)',
15         'props': styles['even']}
16    ], [
17        {'selector': 'tr:nth-child(odd)',
18         'props': styles['odd']}
19    ])
20
21 display(styled_df)
```

aggregere

Aggerger data med pivot

```
1 df_pivot = df.pivot_table(values=['Løpesko',
2                                     'Fotballsko', 'Tursko'], index='År', aggfunc=sum)
3 display(df_pivot)
4 # Summer over radene
5 s_pivot = df_pivot.sum(axis=1)
6 display(type(s_pivot),s_pivot)
```

✓ 0s

År	Fotballsko	Løpesko	Tursko
	Nord	Sør	Øst
2020	390	265	530
2021	440	310	640
2022	480	310	650

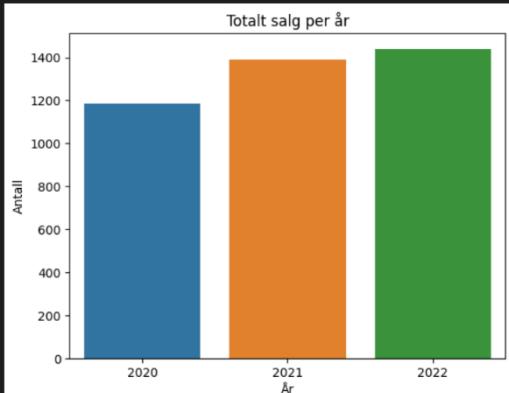
pandas.core.series.Series

```
År
2020    1185
2021    1390
2022    1440
dtype: int64
```

Vis totalt salg per år i et søylediagram

```
1 sns.barplot(x=s_pivot.index, y=s_pivot.values)
2 plt.title("Totalt salg per år")
3 plt.ylabel("Antall")
4 plt.xlabel("År")
5 plt.show()
```

✓ 0s



Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Kombiner tabeller med concat og merge

Når vi arbeider med flere tabeller, kan vi bruke `concat` og `merge` for å kombinere og analysere dataene på en mer omfattende måte. Vi har allerede lært å bruke `append` for å legge til elementer i lister, tupler, ordbøker og sett. Når vi ønsker å skjøte to eller flere tabeller av typen `DataFrame`, bruker vi i stedet `concat`. Så lenge tabellene har tilsvarende kolonner eller rader, kan vi skjøte tabeller både

horisontalt og vertikalt. Derfor har vi også brukt `concat` når vi har lagt en ny rad til en `DataFrame`. Vi starter eksempelet med en (2,3) `DataFrame` som inneholder kolonnene `Frukt`, `Antall` og `Vekt` samt to rader (eple og banan), se filen `3_8_4_concat_og_merge.ipynb`.

Deretter oppretter vi en tilsvarende `DataFrame` med kiwi og appelsin. Tabellene har tilsvarende kolonner og kan skjøtes vertikalt med `concat`. `axis=0` er standard og trengs ikke oppgis. Etter vi har skjøtet tabellene er indeksen 0,1,0,1, så vi trenger å reindeksere tabellen for at indeksen skal bli 0,1,2,3.

Vi ønsker nå å legge til et par nye kolonner, en for farge og en annen for pris. Først oppretter vi en (4,2) `DataFrame` med kolonnene `Farge` og `Pris` og som har 4 rader for henholdsvis eple, banan, kiwi og appelsin. Tabellene har nå tilsvarende rader og kan skjøtes horisontalt med `concat` og `axis=1`.

### Kombinere tabeller med concat og merge

#### concat

Opprett en DataFrame

```
1 import pandas as pd
2 df1 = pd.DataFrame({'Frukt': ['eple', 'banan'],
3                      'Antall': [5, 7],
4                      'Vekt': [2.1, 1.3]})
5 display(df1)
6
```

✓ 0.0s

	Frukt	Antall	Vekt
0	eple	5	2.1
1	banan	7	1.3

#### Legg til rader (vertikalt)

```
1 df2 = pd.DataFrame({'Frukt': ['kiwi', 'appelsin'],
2                      'Antall': [4, 3],
3                      'Vekt': [0.9, 0.8]})
4 display(df2)
5 df3 = pd.concat([df1, df2], axis=0) # axis=0 er standard
6 df3 = df3.reset_index(drop=True) # erstatt gammel index med ny
7 display(df3)
```

✓ 0.0s

	Frukt	Antall	Vekt
0	kiwi	4	0.9
1	appelsin	3	0.8
0	eple	5	2.1
1	banan	7	1.3

#### Legg til kolonner (horisontalt)

```
1 df4 = pd.DataFrame({'Farge': ['rød', 'gul', 'grønn', 'oransje'],
2                      'Kilopris': [32, 25, 41, 28]})
3 display(df4)
4
```

✓ 0.0s

	Farge	Kilopris
0	rød	32
1	gul	25
2	grønn	41
3	oransje	28

```
1 df5 = pd.concat([df3, df4], axis=1)
2 display(df5)
3
```

✓ 0.0s

	Frukt	Antall	Vekt	Farge	Kilopris
0	eple	5	2.1	rød	32
1	banan	7	1.3	gul	25
2	kiwi	4	0.9	grønn	41
3	appelsin	3	0.8	oransje	28

Husk også at vi tidligere har lært å opprette en ny kolonne ved å bruke andre kolonner i utregninger.

```
1 df5['Pris'] = df5['Vekt'] * df5['Kilopris']
2 display(df5)
3
```

✓ 0.0s

	Frukt	Antall	Vekt	Farge	Kilopris	Pris
0	eple	5	2.1	rød	32	67.2
1	banan	7	1.3	gul	25	32.5
2	kiwi	4	0.9	grønn	41	36.9
3	appelsin	3	0.8	oransje	28	22.4

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

`merge` kombinerer også tabeller horisontalt på en lignende måte som `concat`, men det er noen forskjeller. Mens `concat` krever at tabellene har likt antall rader, kan `merge` kombinere tabeller selv om antallet rader er ulikt. Dette skyldes at `merge` bruker en felles kolonne for å koble sammen radene i tabellene. Det finnes ulike måter å koble tabellene på, som `inner`, `outer`, `left` og `right`, som bestemmer hvordan radene skal kombineres i den nye tabellen.

Vi begynner eksempelet med å opprette to tabeller (`DataFrame`) som har en felles kolonne **Frukt**. I den ene tabellen har vi tre rader med **Eple**, **Pære** og **Banan**, og i den andre tabellen har vi bare to rader med **Eple** og **Banan**. Ved en indre sammenføying (`how='inner'`) får vi bare med radene hvor verdiene i felleskolonnen (`on=Frukt`) er like.

Ved en ytre sammenføying (`how='outer'`) får vi med alle radene fra begge tabellene. Manglende verdier blir fylt med `NaN`.

Ved en venstre sammenføying (`how='left'`) får vi

Venstre sammenføying: Alle radene fra venstre tabell og kun rader fra høyre tabell som matcher

```
1 df_left = pd.merge(df1, df2, on='Frukt', how='left')
2 display(df_left)
✓ 0.0s
```

Frukt	Antall	Pris
Eple	10	5.0
Pære	20	NaN
Banan	30	10.0

med alle radene fra

den venstre tabellen og kun rader som matcher fra den høyre tabellen. Manglende verdier blir fylt med `NaN`.

Ved en

høyre sammenføying (`how='right'`) får vi med alle radene fra den høyre tabellen og kun rader som matcher fra den venstre tabellen. Manglende verdier blir fylt med `NaN`.

Opprett datasettene

```
1 df1 = pd.DataFrame({'Frukt': ['Eple', 'Pære', 'Banan'],
2                      'Antall': [10, 20, 30]})
3 df2 = pd.DataFrame({'Frukt': ['Eple', 'Banan'],
4                      'Pris': [5, 10]})
5 display(df1)
6 display(df2)
7
```

Frukt	Antall
Eple	10
Pære	20
Banan	30

Frukt	Pris
Eple	5
Banan	10

Indre sammenføying: Verdiene må finnes i begge tabellene

```
1 df_inner = pd.merge(df1, df2, on='Frukt', how='inner')
2 display(df_inner)
✓ 0.1s
```

Frukt	Antall	Pris
Eple	10	5
Banan	30	10

Ytre sammenføying: Alle rader fra begge tabellene

```
1 df_outer = pd.merge(df1, df2, on='Frukt', how='outer')
2 display(df_outer)
✓ 0.0s
```

Frukt	Antall	Pris
Eple	10	5.0
Pære	20	NaN
Banan	30	10.0

den venstre tabellen og kun rader som matcher fra den høyre tabellen. Manglende verdier blir fylt med `NaN`.

Høyre sammenføying: Alle radene fra høyre tabell og kun rader fra venstre tabell som matcher

```
1 df_right = pd.merge(df1, df2, on='Frukt', how='right')
2 display(df_right)
✓ 0.0s
```

Frukt	Antall	Pris
Eple	10	5
Banan	30	10

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### iterrows

`iterrows()` gir oss en måte å iterere gjennom radene i en `DataFrame`, spesielt nyttig når vi håndterer komplekse datatyper eller utfører operasjoner på deler av dataene. Vi har tidligere lært om `for`- og `while`-løkker og sett at vi kan gå gjennom en endimensjonal liste (*iterable* objekt) med eksempelvis `for person in personer`:

Når vi skal gå gjennom radene i en

`DataFrame`, må vi imidlertid bruke `iterrows()` på formen `for index, row in df.iterrows():` `index` er et heltall og `row` er en `Series`.

`iterrows` kan også være nyttig når vi har en `DataFrame` med komplekse datatyper, for eksempel når vi har en kolonne med lister eller en ordbok (`dict`), og vi vil utføre operasjoner på deler av disse datatypene. I dette eksempelet begynner vi med en `DataFrame` som har to kolonner, `Navn` og `Telefon`, hvor verdiene til `Telefon` er en ordbok med hjemme- og mobiltelefonnummer, se filen `3_8_5_iterrows.ipynb`.

Vi ønsker å hente ut hjemme- mobiltelefonnumrene å ha dem i egne kolonner. Først oppretter vi de to kolonnene, `Hjemme` og `Mobil`.

Så bruker vi `iterrows()` for å få tak i numrene og setter disse verdiene inn i riktig kolonne i riktig rad (`index`) med `df.at`. Til slutt sletter vi den opprinnelige Telefon-kolonnen som inneholdt ordbøkene.

### Ta med minst dette

I denne bolken har vi gått gjennom avanserte temaer i pandas, som håndtering av manglende data, komplekse beregninger med `apply()`, omforming av tabeller med `melt` og `pivot`, og kombinering av tabeller med `concat` og `merge`. Vi har også sett på `iterrows()` for å iterere gjennom rader i en `DataFrame`.

### Øvingsoppgaver

#### 3.8.1

Bruk datasettet `exercise` som følger med Seaborn

```
import seaborn as sns  
df = sns.load_dataset('exercise')
```

- Skriv ut rad (indeks) 33, 34 og 25 fra datasettet
- Skriv ut `pulse`, `time` og `kind` til de to radene som har høyest `pulse`

Opprett en DataFrame med komplekse data

```
1 import pandas as pd  
2  
3 # Oppretter DataFrame  
4 data = {'Navn': ['Per', 'Kari', 'Ola'],  
5         'Telefon': [{['hjemme': '12345678', 'mobil': '99887766'},  
6                      {'hjemme': '87654321', 'mobil': '11223344'},  
7                      {'hjemme': '56789012', 'mobil': '44332211'}]}  
8  
9 df = pd.DataFrame(data)  
10 display(df)
```

0.8s

Navn	Telefon
Per	{'hjemme': '12345678', 'mobil': '99887766'}
Kari	{'hjemme': '87654321', 'mobil': '11223344'}
Ola	{'hjemme': '56789012', 'mobil': '44332211'}

Lag nye kolonner

```
1 df['Hjemme'] = ''  
2 df['Mobil'] = ''  
3 display(df)
```

0.0s

Navn	Telefon	Hjemme	Mobil
Per	{'hjemme': '12345678', 'mobil': '99887766'}		
Kari	{'hjemme': '87654321', 'mobil': '11223344'}		
Ola	{'hjemme': '56789012', 'mobil': '44332211'}		

Bruk iterrows til å hente telefonnumrene

Legg de inn i respektive kolonne

Slett til slutt den opprinnelige kolonnen

```
1 for index, row in df.iterrows():  
2     hjemme = row['Telefon'][['hjemme']]  
3     df.at[index, 'Hjemme'] = hjemme  
4     mobil = row['Telefon'][['mobil']]  
5     df.at[index, 'Mobil'] = mobil  
6  
7 df.drop('Telefon', axis=1, inplace=True)  
8 display(df)
```

0.0s

Navn	Hjemme	Mobil
Per	12345678	99887766
Kari	87654321	11223344
Ola	56789012	44332211

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 3.8.2

Bruk datasettet `flights` som følger med Seaborn

```
df = sns.load_dataset('flights')
```

- Skriv ut totalt antall passasjerer på 1950-tallet (1950-1959)
- Lag en graf som viser antall passasjerer per år i årene 1953 til 1957

### 3.8.3

Bruk vedlagte datasettet `tips_med_na.csv`.

`shape`-funksjonen finner antall rader og kolonner i en `DataFrame`. `shape` returnerer en `tuple` med to verdier: antall rader og antall kolonner.

`axis`-parametren i Pandas bestemmer om funksjonen skal brukes på tvers av radene eller kolonnene. Eksempelvis vil `sum(axis=0)` eller `sum()` summere verdiene i kolonnene, `sum(axis=1)` vil summere verdiene i radene.

- Skriv ut antall rader
  - Skriv ut antall kolonner
  - Skriv ut antall celler
  - Skriv ut antall `NaN`
  - Skriv ut hvor mange % data som mangler
- Skriv ut hvor mange `NaN` det er i hver kolonne
- Skriv ut radene som inneholder `NaN`
- Hvor mange prosent data mister vi hvis vi fjerner alle radene med `NaN`?

### 3.8.4

Vis et linjediagram med antall kvinner fra det demokratiske og det republikanske parti i USAs kongress for alle periodene i det vedlagte datasettet

`women_in_the_house_of_representatives.csv`

### 3.8.5

Opprett to tabeller (`DataFrame`) som vist nedenfor og føy dem sammen. Begrunn valget av koblingstype. Mangledo data kan legges in som `numpy.nan`.

df1	
Navn	Alder
Erik	18
Frida	19
Gustav	17
Hanne	19

df2	
Navn	Bosted
Erik	Askim
Gustav	
Hanne	Mysen
Ivar	Moss

## 3.9 Reelle datasett med OOP og GUI

### Bolkens innhold

Innledning .....	222
Bokstaver med lik bredde i tabellvisning.....	223
Seaborn-diagram i tkinter-vindu .....	224
Tkinter App-GUI mal .....	224
Ta med minst dette.....	226
Øvingsoppgaver.....	227

### Kompetansemål:

- utforske og vurdere alternative løsninger for design og implementering av et program
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer
- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

### Innledning

Vi har tidligere lært å skrive ut en `pandas.DataFrame` med `print`, som viser tabellen i terminalvinduet. En ulempe med denne metoden er at vi noen ganger må endre standardinnstillingene for å få med alle radene, eller forstørre vinduet for å se alle kolonnene. Det blir penere med `display` i *Jupyter Notebooks*. Når vi skal vise diagrammer, har vi brukt `seaborn`, som åpner et `matplotlib`-vindu. Dette har vært statiske, prosedyreorienterte programmer.

Udir<sup>25</sup> har antydet at "*informasjon fra datasett knyttes sammen med objektorientert utvikling og programmering samt utvikling av dynamiske løsninger med brukergrensesnitt*". Det første eksempelet på dette kom i eksamensoppgave 9, våren 2024, se [Øvingsoppgave 3.9.4](#). I denne bolken syr vi sammen noe av det vi har lært til nå for å oppnå nettopp dette. Selv om det foreløpig ikke har vært sagt noe om grafiske brukergrensesnitt i denne sammenheng, har vi valgt å anvende `tkinter`.

<sup>25</sup> Se [Eksamensveiledning Vår 2024](#)

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Bokstaver med lik bredde i tabellvisning

Da vi skulle skrive ut innholdet av en ordbok i et tkinter-vindu, gjorde vi det med en løkke hvor vi la ut *Label-widgeteter* med *grid*, se [2.2 Tupler og ordbøker](#). Vi så også på andre alternativer som *Treeview* og *PandasTable*, som oppretter sitt eget vindu. Her skal vi bruke et annet alternativ, nemlig å skrive ut hele tabellen til ett tekstfelt.

navn	alder
Knut	32
Tiril	28
Anne	41
Oscar	35

For at teksten skal være riktig justert, må vi passe på å bruke en skriftype hvor alle bokstavene har samme bredde (*monospaced font*).

```
import tkinter as tk
import pandas as pd
import platform
personer = [
    {'navn': 'Knut', 'alder': 32},
    {'navn': 'Tiril', 'alder': 28},
    {'navn': 'Anne', 'alder': 41},
    {'navn': 'Oscar', 'alder': 35}
]

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Bokstaver med fast bredde")
        self.root.geometry('325x200')
        # Bruk skriftype med fast bredde på alle plattformer
        if platform.system() == "Windows":
            font = ('Consolas', 18)
        elif platform.system() == "Darwin": # macOS
            font = ('Menlo', 18)
        else: # Linux
            font = ('Monospace', 18)
        # Sett standard skriftype for alle tkinter-widgeteter
        self.root.option_add('*Font', font)
        df = pd.DataFrame(personer)
        # Ikke vis indekskolonnen
        tekst = df.to_string(index=False)
        text_widget = tk.Text(self.root, wrap='none', padx=70, pady=10)
        text_widget.insert('1.0', tekst)
        # Gjør tekstfeltet skrivebeskyttet
        text_widget.configure(state='disabled')
        text_widget.pack(padx=10, pady=10)

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = App()
    app.run()
```

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Seaborn-diagram i tkinter-vindu

I boklene [1.9](#) og [2.9](#) lærte vi å presentere diagrammer laget med seaborn i **matplotlib vinduer** ved å bruke `plt.show()`. I bokene [3.1](#) og [3.2](#) så vi hvordan vi kan vise diagrammer i Jupyter-notatbøker, samt hvordan vi kan lagre dem som **HTML-websider** og **PDF-dokumenter**. Nå skal vi lære å presentere seaborn-diagrammer i et **tkinter-vindu**.

Når vi ønsker å vise et seaborn-diagram i en tkinter app, kan vi bruke klassen

`FigureCanvasTkAgg` for å opprette en tkinter *widget* som kalles et lerret. Denne *widgeten* kan deretter inkluderes i et tkinter vindu for å vise diagrammet. `FigureCanvasTkAgg` er en del av modulen `matplotlib.backends.backend_tkagg` og må importeres.

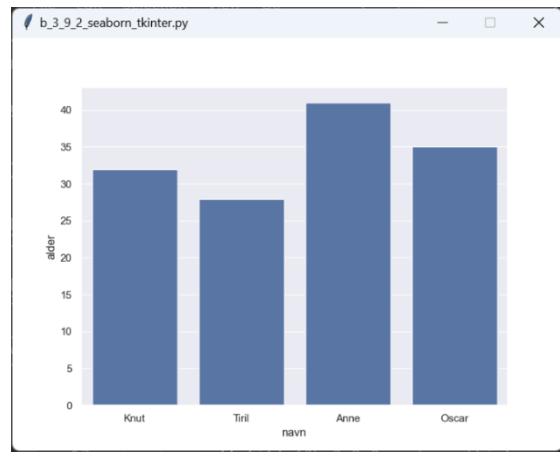
Vi bytter ut `plt.show()` med

```
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack()
canvas.draw()
```

For at applikasjonen skal avslutte korrekt, må vi kalle `plt.close('all')` og `destroy()`. Hvis vi oppretter `tk.Tk` som en komposisjon (dvs. ved å ha et `tk.Tk`-objekt som et attributt i klassen, f.eks. `self.root = tk.Tk()`), må vi bruke `self.root.protocol("WM_DELETE_WINDOW", self.on_closing)` og gjøre dette fra `on_closing`-metoden. Her har vi imidlertid valgt å arve fra `tk.Tk`, og da holder det med å overskrive `destroy()`-metoden. Se [Embedding in Tk](#).

### Tkinter App-GUI mal

Når vi skal både analysere data med pandas og presentere resultatene med tkinter i samme program, kan det være smart å skille GUI-koden fra applikasjonslogikken. Å [skille mellom oppgaver](#) (*Separation of concerns*) er et kjent designprinsipp innen informatikk. Et typisk eksempel på dette er HTML, CSS og JavaScript i webutvikling. Tidligere var alt samlet i HTML-filer, men ved å separere stil (CSS) og funksjonalitet (JavaScript) fra innhold (HTML) blir koden bedre organisert, lettere å lese, mer vedlikeholdsvennlig og enklere å gjenbruke. Se også [MVC](#), [5.8 Designmønstre](#)



```
b_3_9_2_seaborn_tkinter.py

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

df = pd.DataFrame({
    'Navn': ['Knut', 'Tiril', 'Anne', 'Oscar'],
    'Alder': [32, 28, 41, 35]
})

class App(tk.Tk):
    def __init__(self):
        '''Vise et Seaborn-diagram i et tkinter-vindu'''
        super().__init__()
        self.title("b_3_9_2_seaborn_tkinter.py")
        self.geometry("800x600")
        self.resizable(False, False)
        sns.set_theme()
        fig, ax = plt.subplots(figsize=(9, 6))
        sns.barplot(data=df, x='Navn', y='Alder')
        fig.suptitle('Seaborn-diagram i tkinter-vindu',
                     fontsize=20)
        ax.set_title('Aldersfordeling')
        plt.tight_layout()
        canvas = FigureCanvasTkAgg(fig, master=self)

    def run(self):
        '''Start hendelsessløyfen og hold vinduet åpent'''
        self.mainloop()

    def destroy(self) -> None:
        ...
        Lukk alle matplotlib-figurer og avslutt vinduet.
        Nødvendig når vi bruker matplotlib i tkinter.
        ...
        plt.close('all')
        super().destroy()

if __name__ == "__main__":
    ...
    Start applikasjonen hvis filen kjøres som et skript.
    Kjøres ikke hvis denne filen importeres som modul.
    ...
    app = App()
    app.run()
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Vi tar utgangspunkt i [den objektorienterte malen for tkinter](#) som ble presentert i 2.1 Tupler og ordbøker. For app-instansen gjelder

- Det hele settes i gang som tidligere med `app=App()` og `app.run()`.
- `App`-konstruktøren oppretter en `Gui`-instans og henter datagrunnlaget med `get_data()`
- `run()` starter hendelsessløyfen og holder tkinter vinduet åpent
- `handle_request()` blir kalt fra `Gui`-instansen, behandler data og ber `Gui`-instansen om å vise resultatet

I `Gui`-klassen vil vi nå implementere både tabell- og diagramvisning som forklart i de to foregående avsnittene. Konstruktøren til `Gui`-klassen tar først vare på en referanse til `App`-instansen for å kunne kalle metoder i den klassen. Deretter settes vinduet opp sentrert på skjermen og det velges en *monospaced font* for alle *widget*-ene. Til slutt opprettes et kontrollområde (`Frame`) for valg av visning og et visningsområde (`Frame`) for resultatene, enten der er en tabell eller et diagram.

I kontrollområdet har vi kun to knapper, men her kan vi bruke radioknapper, avhukingsbokser, liste- eller combobokser, avhengig av oppgaven krever. Det er også grunnen til at vi ikke kaller `app.handle_request()` direkte fra `command`-argumentet i knappene. Det kan hende vi ønsker å sende med flere argumenter, og det blir fort ulesbart med `lambda`-funksjoner.

```
class App:  
    def __init__(self):  
        '''Opprett en Gui-instans og hent data'''  
        self.gui = Gui(self)  
        self.df = self.get_data()  
  
    def get_data(self):  
        '''Returner data, gjerne fra en fil'''  
        return pd.DataFrame({  
            'Navn': ['Knut', 'Tiril', 'Anne', 'Oscar'],  
            'Alder': [32, 28, 41, 35]  
        })  
  
    def handle_request(self, request):  
        '''Behandle forespørsmålet og se om gui skal vise resultatet'''  
        if request == 'show_table':  
            data = self.get_data().sort_values('Alder', ascending=False)  
            data = data.to_string(index=False)  
            data = f"Personer sortert etter alder:\n\n{data}"  
            self.gui.display_text(data)  
        elif request == 'show_graph':  
            data = self.get_data().copy()  
            self.gui.display_graph(data, 'Aldersfordeling')  
  
    def run(self):  
        '''Start hendelsessløyfen og hold vinduet åpent'''  
        self.gui.mainloop()
```

```
class Gui(tk.Tk):  
    def __init__(self, app):  
        '''Opprett GUI-vindu med kontroll- og visningsområder'''  
        super().__init__()  
        # Lagre en referanse til app-instansen  
        self.app = app  
        # Vindustittel, størrelse og sentrering  
        self.title("Tkinter OOP App GUI Mal")  
        b, h = 500, 400  
        bs, hs = self.winfo_screenwidth(), self.winfo_screenheight()  
        self.geometry(f'{b}x{h}+{bs//2-b//2}+{hs//2-h//2}')  
        self.resizable(False, False)  
        # Bruk skriftype med lik bredde på alle plattformer  
        # For at tabeller skal vises pent med Text-widget  
        if platform.system() == "Windows":  
            font = ('Consolas', 18)  
        elif platform.system() == "Darwin": # macOS  
            font = ('Menlo', 18)  
        else: # Linux  
            font = ('Monospace', 18)  
        self.option_add('*Font', font)  
        # Kontrollområde for valg av visning  
        self.controls_frame = tk.Frame(self, relief='raised', bd=1)  
        self.controls_frame.pack(side=tk.TOP, fill='x')  
        self.setup_controls()  
        # Visningsområde  
        self.display_frame = tk.Frame(self)  
        self.display_frame.pack(side=tk.RIGHT, fill='both', expand=True)
```

```
def setup_controls(self):  
    '''Opprett kontroller for valg av visning'''  
    tk.Button(self.controls_frame, text='Vis Tabell',  
              command=self.show_table_request).grid(  
        row=0, column=0, padx=10, pady=10)  
    tk.Button(self.controls_frame, text='Vis Graf',  
              command=self.show_graph_request).grid(  
        row=0, column=1, padx=10, pady=10)  
  
def show_table_request(self):  
    ...  
    Be app om å behandle forespørsmålet,  
    som deretter ber gui om å vise resultatet i en tabell  
    ...  
    self.app.handle_request('show_table')  
  
def show_graph_request(self):  
    ...  
    Be app om å behandle forespørsmålet,  
    som deretter ber gui om å vise resultatet i en graf  
    ...  
    self.app.handle_request('show_graph')
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

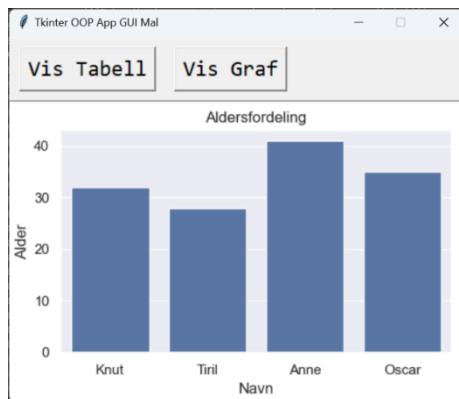
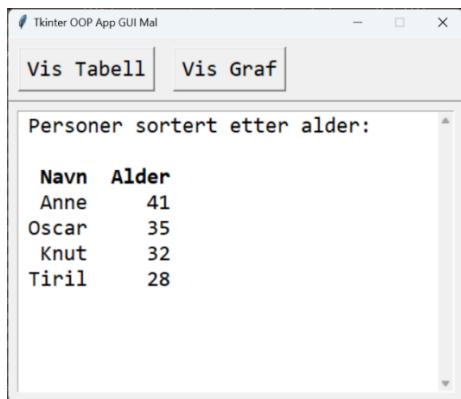
`display_text()` har vi fiffet opp noe fra første avsnitt. Vi har lagt til et rullefelt, en tittel og blank linje før tabellen (i `handle_request`), samt formatert teksten med marger og kolonneoverskriftene (tredje linje) med fet skrift

```
def display_text(self, data):
    '''Vis tekst i visningsområdet'''
    self.clear_display_frame()
    # Tekstfelt
    output = tk.Text(self.display_frame, wrap='word')
    output.pack(padx=10, pady=10, fill='both', expand=True)
    # Rullefelt
    scroll = ttk.Scrollbar(output, orient='vertical',
                           command=output.yview)
    scroll.pack(side='right', fill='y')
    output['yscrollcommand'] = scroll.set
    # Tag for marger
    output.tag_configure(
        "marg", lmargin1=10, lmargin2=10, rmargin=10)
    # Tag for fet skrift
    font = Font(output, output.cget("font"))
    font.config(weight="bold")
    output.tag_configure('fet', font=font)
    # Sett inn tekst med fete kolonnenavn
    output.insert("1.0", data, "marg")
    output.tag_add("fet", "3.0", "3.end")
    output['state'] = 'disabled'
```

`display_graph()` fungerer som forklart i forrige avsnitt, men vi tømmer først visningsområdet for `widget`-er med `clear_display_frame()`.

```
def display_graph(self, data, title):
    '''Vis graf i visningsområdet'''
    self.clear_display_frame()
    sns.set_theme()
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.barplot(data=data, x='Navn', y='Alder', ax=ax)
    ax.set_title(title)
    plt.tight_layout()
    canvas = FigureCanvasTkAgg(fig, master=self.display_frame)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

def clear_display_frame(self) -> None:
    '''Fjern alle widgeter fra visningsområdet (før ny visning)'''
    for widget in self.display_frame.winfo_children():
        widget.destroy()
```



### Ta med minst dette

Vi kan skrive ut `pandas.DataFrame`-tabeller til en `Text-widget` ved å bruke `pd.DataFrame.to_string(index=False)` og en `monospaced font`. Med `FigureCanvasTkAgg` fra `matplotlib.backends.backend_tkagg` kan vi integrere Seaborn-diagrammer i et Tkinter-vindu. Når vi bruker `matplotlib` i Tkinter, må vi sikre at programmets vindu lukkes korrekt ved å kalle `plt.close('all')` og `destroy()`-metoden. Ved å skille GUI-kode fra applikasjonslogikk blir koden bedre organisert, lettare å lese, vedlikeholde og gjenbruke. Denne bolken presenterer en mal som implementerer disse prinsippene.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Øvingsoppgaver

#### 3.9.1

**Iris**-datasettet er et kjent datasett i maskinlæring og statistikk, som inneholder informasjon om 150 iris-blomster fordelt på tre arter: *Setosa*, *Versicolor* og *Virginica*. For hver blomst er det målt fire egenskaper:

- *Sepal length* (begerbladets lengde)
- *Sepal width* (begerbladets bredde)
- *Petal length* (kronbladets lengde)
- *Petal width* (kronbladets bredde)

Lag et objektorientert program hvor brukeren kan velge mellom tre visninger:

1. Datasettet (hele tabellen)
2. Et stolpediagram med gjennomsnittlig *sepal length* per art.
3. Et kakediagram over antall blomster i datasettet per art.

Du kan lese inn datasettet med `sns.load_dataset('iris')`, som krever internettilkobling, eller benytte vedlagte **iris.csv** eller **iris.json**.

#### 3.9.2

**Titanic**-datasettet er et kjent datasett som inneholder informasjon om et utvalg av passasjerene på det skjebnesvandre skipet RMS Titanic. For hver passasjer er det registrert flere opplysninger, blant annet

- *survived* (overlevde: 0 = omkom, 1 = overlevde)
- *pclass* (klasse)
- *sex* (kjønn)
- *age* (alder)
- *fare* (billettpris i £)
- *embark\_town* (avreisehavn)

Lag et objektorientert program hvor brukeren kan velge mellom tre visninger:

1. Brukeren skal kunne velge passasjerklassen fra en *combobox* (Alle, 1, 2, 3)  
Programmet skal deretter vise en tabell med passasjerene i den valgte klassen.
2. Brukeren skal kunne velge mellom to ulike diagrammer ved hjelp av radioknapper:
  - Et stolpediagram som viser gjennomsnittlig billettpris per avreisehavn
  - Et sektordiagram som viser fordelingen av overlevende og omkomne passasjerer

Du kan lese inn datasettet med `sns.load_dataset('titanic')`, som krever internettilkobling, eller benytte vedlagte **titanic.csv** eller **titanic.json**.

## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

### 3.9.3

Datasettet **aksjekurser.csv** inneholder slutt kurser for fem selskaper for hver dag i 2023: Apple (AAPL), Alphabet (GOOGL), Amazon (AMZN), Microsoft (MSFT) og Meta (META). Datasettet har tre kolonner:

- *Date* (Dato for slutt kurseren)
- *Company* (Selskapets ticker)
- *Close* (Slutt kurs for dagen)

Lag et objektorientert program hvor brukeren kan velge mellom to visninger:

1. Brukeren skal kunne velge et selskap fra en *combobox*. Programmet skal deretter vise en tabell med slutt kursene for det valgte selskapet.
2. Brukeren skal kunne velge flere selskaper ved hjelp av avhukningsbokser. Programmet skal deretter vise et linjediagram som viser utviklingen av slutt kursene for de valgte selskapene.

### 3.9.4

Analyse av tidsbruk: Oppgave 9, [IT-2 Eksamens Vår 2024](#).

#### **Vedlegg 2 – datasett med tidsbruk på ulike aktiviteter**

Vedlagt ligger det et datasett som inneholder en oversikt over tidsbruk på ulike aktiviteter, i CSV- og JSON-format. Du kan velge hvilket av formatene du vil bruke. Datasettet kan lastes ned ved [å følge denne lenken](#).

Datasettet inneholder tid brukt til ulike aktiviteter en gjennomsnittsdag året 2000 kategorisert etter aktiviteter og kjønn.

De ulike aktivitetene er gruppert i kategorier hvor tidsbruken er summert for kategoriene. Aktivitetene under hver kategori er kodet med innrykk.

Tidsbruk er angitt i hele timer og minutter skilt med punktum. *Merk at dette ikke er timer med desimaltall.*

*Datasettet er kodet med tegnsettet UTF-8. Skilletegnet i CSV-filen er semikolon.*

Last ned datasettet i ønsket format og gjør deg kjent med det. Du må også forberede deg på å behandle datasettet med programmering og tilpasse data i settet for å kunne gjøre de aktuelle beregningene. I tillegg må du kontrollere at du har de nødvendige ressursene installert på den datamaskinen du skal bruke på eksamen. På eksamenen vil du få to oppgaver hvor du skal behandle dette datasettet og presentere informasjon fra det med programmering.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

**Oppgave 9 – analyse av tidsbruk**

I denne oppgaven skal du bruke datasettet fra forberedelsen. Hvis du ikke har forberedt dette, kan du laste ned datasettet fra vedlegg 2 nå.

**a) Presentasjon av data**

Lag et program som presenterer dataene fra datasettet i en tabellignende visning med følgende kolonner: «Aktivitet», «Kjønn» og «Tidsbruk».

**b) Filtrering etter kjønn**

Utvid programmet slik det skal være mulig å vise et utvalg av dataene i tabellen fra punkt a etter kjønn. Man skal kunne velge mellom «Alle», «Menn» og «Kvinner», og når kjønnet er valgt, skal visningen oppdateres med aktuelle data for det valgte kjønnet.

**c) Visualisering av data**

Utvid programmet og lag to diagrammer:

- et stolpediagram som viser fordelingen av tidsbruk på aktivitetene for det valgte kjønnet – hver aktivitet skal være representert med en stolpe, og høyden på stolpen skal vise tidsbruken for aktivitetene for det valgte kjønnet i punkt b
- et sektordiagram som viser andelen tid av døgnet brukt på hver kategori av aktiviteter for det valgte kjønnet i punkt b

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

### 3.10 Kunstig intelligens

#### Bolkens innhold

Fra idé til virkelighet.....	230
Verktøy .....	231
Systemutvikling .....	232
Muligheter, utfordringer og konsekvenser.....	233
Etiske dilemmaer.....	234
Ta med minst dette.....	235
Øvingsoppgaver.....	235

#### Kompetanse mål:

- utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger
- drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn

#### Fra idé til virkelighet

Kunstig intelligens (AI, *Artificial intelligence*) er dataprogrammer som kan utføre oppgaver som normalt krever menneskelig intelligens, men hva er menneskelig intelligens? Det er lettere spurt enn svart, for intelligens er et sekkeord og det finnes flere definisjoner. Intelligens omfatter evner som å lære, forstå og bruke informasjon, og det forekommer i ulike former, som språklig, logisk, visuell, emosjonell og sosial intelligens. Derfor er det fortsatt debatt om hva intelligens egentlig er. Vi begrenser oss til å registrere at datamaskiner har en annen type intelligens enn mennesker. Datamaskiner er gode på å behandle store mengder data veldig raskt og utføre gjentakende oppgaver, som å gjenkjenne mønstre i data. Vi er eksempelvis flinkere til å tenke kreativt, bruke følelser og forstå sosiale signaler.

Utviklingen av kunstig intelligens begynte på 1950-tallet, med publikasjoner av forskere som John McCarthy, Marvin Minsky, Claude Shannon og Alan Turing. Sistnevnte er blant annet kjent for *Turingtesten* (*An Imitation Game*). Denne testen går ut på å stille spørsmål til både et menneske og en maskin uten å vite hvem som er hvem, og hvis den som spør ofte tror at det er mennesket som svarer når det i virkeligheten er maskinen, har maskinen bestått testen. Etter oppstarten var det stor fremgang, optimisme og høye forventninger. Det viste seg at forventningene



## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

rundt AI var for høye. Interessen dabbet av og investeringene uteble, noe som førte til en periode kalt [AI-vinteren](#) på 1980-tallet. Denne perioden var preget av en generell skuffelse over teknologien, en nedgang i aktivitet og reduserte bevilgninger.

Etter AI-vinteren kom en ny periode med ny begeistring og økte investeringer i kunstig intelligens. Internett har spilt en stor rolle i denne sammenheng, for det ble mulig å samle inn og lagre store mengder data, noe som har ført til en økning i treningen av AI-modeller. Sammen med kraftigere datamaskiner, bedre algoritmer og utviklingen av dyplæring<sup>26</sup>, fikk AI en renessanse. Bruken av AI i mange bransjer og områder nærmest eksploderte. [ChatGPT](#) er en AI-basert tekstgenereringsmodell som bruker nevrale nettverk til å svare basert på læring fra trente data. [Ifølge en analyse](#) fra dataselskapet Similarweb, nådde ChatGPT 100 millioner unike besøkende i januar 2023, bare to måneder etter lanseringen. Dette representerte den raskeste veksten for en internettapp noensinne. Til sammenligning, tok det TikTok 9 måneder og Instagram to år å nå 100 millioner brukere. AI blir stadig mer integrert i samfunnet og gjør seg gjeldene i sektorer som helsevesen, finans, transport og teknologi. Samtidig blir det tydeligere at AI kan føre til uønskede konsekvenser, og i [Nasjonal strategi for kunstig intelligens](#) (2020) foreslår regjeringen at tilsynsmyndigheter skal føre kontroll med bruken av AI.



### Verktøy

AI er ikke én teknologi, men et bredt fagområde hvor *Machine Learning (ML)* står sentralt. ML kan deles inn i tre hovedkategorier:

**Veiledet læring (SL, Supervised Learning)** hvor algoritmen lærer fra en trent datamengde med kjent utgangspunkt for å forutsi utfall for nye data.

**Ikke-veiledet læring (UL, Unsupervised Learning)** hvor algoritmen lærer å identifisere mønstre i data uten å ha noen kjent utgangspunkt eller mål.

**Forsterket læring (RL, Reinforcement Learning)** hvor algoritmen lærer ved å utføre handlinger og få belønning eller straff for sine valg.

Innenfor veiledet læring finner vi blant annet

**Naturlig språkprosessering (NLP, Natural Language Processing)** - som fokuserer på å analysere, forstå og generere menneskelig språk.

**Datasyn (CV, Computer Vision)** som dreier seg om å analysere og forstå digitale bilder og videoer.

**Ansiktsgjenkjenning (FR, Facial Recognition)** som handler om å gjenkjenne menneskers ansikter i digitale bilder og videoer.

<sup>26</sup> Dyplæring er avanserte nevrale nettverk.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

**Kunstige nevrale nettverk** (ANN, Artificial Neural Networks) som simulerer funksjonaliteten til menneskelig hjerne ved å bruke et kunstig nettverk av neuroner.

**Dyplæring** (DL, Deep Learning) - som trenger dypt ned i datastrukturer ved å bruke flere lag med kunstige neuroner.

Andre verktøy innen AI som ikke er en del av ML inkluderer **regelbaserte systemer** (RBS, Rule Based Systems), **kognitiv modellering** (CM, Cognitive Modeling), og **heuristiske algoritmer** (HA, Heuristics Algorithms). Disse teknologiene bruker faste regler eller erfaringer for å løse AI-relaterte oppgaver, i stedet for å lære fra data. Det kan være nyttig når oppgaven som skal løses trenger spesifikke regler som er bestemt av mennesker, vi trenger en rask løsning eller vi ikke har tilstrekkelig data.

AI løser problemstillinger ved hjelp av en rekke metoder, som logisk bevisførsel, planleggingsalgoritmer, optimeringssøk og statistisk læring. Statistisk læring kan igjen deles inn i ulike metoder, som sannsynlighetsmodeller, regresjonsanalyse, klassifisering og gruppering. Osv. Dette gir en annen inndeling av AI enn områder som naturlig språkprosessering (NLP), ansiktsgjenkjenning (FR), kunstig nevrale nettverk (ANN) og dyp læring (DL). Nå er det kanskje lettere å forstå at AI er ikke én teknologi, men et bredt fagområde.

### Systemutvikling

AI spiller en stadig viktigere rolle i systemutvikling, og har mange praktiske fordeler som kan hjelpe utviklere å arbeide mer effektivt og produktivt.

**Autofullføring:** AI kan bidra til å øke utviklernes produktivitet ved å anbefale mulige kodelinjer, noe som gir en raskere og mer effektiv kodeutvikling.

**Kodegenerering:** AI har evnen til å generere kode automatisk, noe som kan spare utviklere for mye tid og redusere feil.

**Feilsøking:** AI kan diagnostisere og løse feil i koden automatisk, slik at utviklerne kan fokusere på mer komplekse oppgaver.

**Testgenerering:** AI kan hjelpe til med å generere automatiske tester for å sikre at koden oppfører seg som forventet.

**Dokumentasjonsgenerering:** AI kan bidra til å generere teknisk dokumentasjon basert på koden og kommentarer, noe som kan være til stor hjelp i store prosjekter.

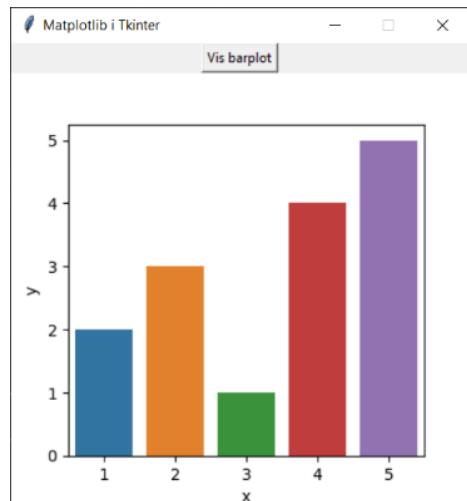
Disse funksjonene er tilgjengelige gjennom en rekke verktøy som Visual Studio Code med IntelliSense, GitHub Copilot- og Codeiumutvidelsene, i andre utviklingsmiljøer og som frittstående tjenester som TabNine, DeepCode og mange flere. Det er viktig å merke seg at AI teknologien fortsatt er i sin spede begynnelse, og at det derfor er nødvendig å være klar over begrensningene.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

ChatGPT, som vi skrev om innledningsvis, er en kunstig intelligens-basert chatbot utviklet av OpenAI. Den benytter seg av maskinlæring (ML), naturlig språkbehandling (NLP) og transformer-teknologi (GPT) for å gi raske og presise svar på brukerne sine spørsmål. ChatGPT kan også hjelpe til med programmering i Python. I skjermkopien til høyre så vi den om å flette to lister i Python, og det gikk greit. Det gjør det ikke alltid, og det kan være grunnen til at "Stack Overflow forbyr kode skrevet av ChatGPT - mener den ofte inneholder feil", se [kode24](#).

I skjermkopien til høyre skrev vi kun tre linjer med kommentarer, så foreslo [GitHub coPilot](#) utvidelsen i VS Code resten (se også [Wikipedia](#)).



Hvis vi skal lage en liste med farger og skriver `farger =`, så kommer det opp forslag til farger. Det kan godkjennes med tabulator-tasten, eller vi kan fortsette å skrive.

```
farger = ["rød", "grønn", "blå", "gul", "lilla", "svart", "hvitt", "rosa", "brun"]
```

### Muligheter, utfordringer og konsekvenser

Kunstig intelligens har potensiale til å påvirke mange bransjer og områder. Her er noen muligheter:

**Helsevesenet:** AI kan brukes i diagnose og behandling av sykdommer, bildeanalyse og utvikling av nye legemidler.

**Finansbransjen:** Det kan anvendes for å forutsi markedstrender, automatisere investeringsbeslutninger og forbedre kundeservice.

Skriv et Python program som fletter listen bokstaver = [a,b,c] med listen tall = [1,2,3]

Her er et Python program for å flette to lister, `bokstaver` og `tall`:

```
SCSS
bokstaver = ['a', 'b', 'c']
tall = [1, 2, 3]

# Bruk zip() funksjonen til å flette lister
resultat = [i for i in zip(bokstaver, tall)]

# Skriv ut resultatet
print(resultat)
```

Output:

```
css
[(a, 1), (b, 2), (c, 3)]
```

```
# Lag et tkinter GUI
# med seaborn barplot
# som vises på canvas

import tkinter as tk
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Lager en klasse for vinduet

class Vindu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Matplotlib i Tkinter")
        self.geometry("400x400")
        self.resizable(False, False)
        self.lag_knapp()
        self.lag_canvas()

    def lag_knapp(self):
        self.knapp = tk.Button(self, text="Vis barplot",
                              command=self.vis_barplot)
        self.knapp.pack()

    def lag_canvas(self):
        self.canvas = tk.Canvas(self, width=400, height=400)
        self.canvas.pack()

    def vis_barplot(self):
        # Lager en dataframe med data
        df = pd.DataFrame({"x": [1, 2, 3, 4, 5], "y": [2, 3, 1, 4, 5]})
        # Lager en barplot med seaborn
        sns.barplot(x="x", y="y", data=df)
        # Viser barplot på canvas med matplotlib
        figure = plt.gcf()
        canvas = FigureCanvasTkAgg(figure, self.canvas)
        canvas.draw()
        canvas.get_tk_widget().pack()

    # Kjører programmet
if __name__ == "__main__":
    # Lager et vindu
    vindu = Vindu()
    # Kjører vinduet
    vindu.mainloop()
```

Vurderingsksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

**E-handel:** Det kan anvendes for å tilpasse anbefalinger til kundene, forbedre søke- og filtreringsfunksjoner, og forbedre leverings- og logistikkprosesser.

**Produksjon:** Kunstig intelligens kan optimere produksjonslinjer, forutsi feil og øke produktiviteten.

**Markedsføring:** Det kan brukes for å skape innhold, forbedre målretting og tilpasse innhold til enkeltbrukere.

**Forsvar:** Det kan overvåke og analysere sikkerhetsrisikoer, samt forbedre beslutningstaking og stridsevner

**Autonome systemer:** AI kan anvendes i selvkjørende biler og skip for å forbedre sikkerheten og effektiviteten.

Kunstig intelligens byr også på en rekke utfordringer som

**Diskriminering:** Hvordan sikre at AI-systemer ikke diskriminerer basert på rase, kjønn, alder eller andre sosiale faktorer?

**Arbeidsledighet:** Hvordan håndtere den økte arbeidsledigheten som følge av automatisering av arbeidsoppgaver?

**Overvåkning:** Hvordan beskytte privatlivet når AI-teknologi blir stadig mer integrert i våre liv og samfunn, og hvor mye overvåkning er akseptabelt?

**Ansvar:** Hvem er ansvarlig hvis AI-systemer gjør feil, og hvordan sikre at de ansvarlige tar ansvar?

**Sikkerhet:** Hvordan beskytte AI-systemer mot angrep og hvordan sikre at de ikke kan brukes til ondsinnede formål?

Noen konsekvenser ved å ta i bruk kunstig intelligens er

**Automatisering:** Bruk av AI fører til økt automatisering av arbeidsprosesser, noe som kan resultere i oppsigelser og økt arbeidsledighet.

**Effektivitet:** AI bidrar til økt effektivitet i mange bransjer, da AI-systemer kan utføre oppgaver raskere og mer nøyaktig enn mennesker.

**Etiske bekymringer:** Bruk av AI kan føre til etiske bekymringer, for eksempel om hvordan personlige data blir håndtert og brukt.

**Regulering:** Bruk av AI krever regulering for å sikre at teknologien blir brukt på en ansvarlig måte og beskytter vår sikkerhet og våre privatliv.

**Økt avhengighet:** Samfunnet blir stadig mer avhengig av AI, noe som fører til økt sårbarhet.

#### **Etiske dilemmaer**

Noen av de mest kontroversielle spørsmålene innen AI inkluderer:

**Databeskyttelse og helse:** Skal vi prioritere beskyttelse av sensitive helseopplysninger eller muligheten til å bruke AI for å forbedre diagnose og behandling av sykdommer?

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

**Autonomi og ansvar:** Skal vi prioritere at AI-systemer har stor grad av autonomi, eller at ansvarlige personer alltid kan identifiseres ved eventuelle ulykker?

**Ærlighet og kommersielle interesser:** Skal vi prioritere at AI-systemer alltid er ærlige og tar hensyn til brukernes beste, eller at de tjener penger for sine eiere?

**Diskriminering og effektivitet:** Skal vi prioritere at AI-systemer er rettferdige og unngår diskriminering, eller at de er så effektive som mulig?

**Kontroll og tillit:** Skal vi prioritere at AI-systemer alltid kan kontrolleres av mennesker, eller at vi bygger tillit til teknologien slik at den kan brukes uten bekymringer?

**Privatliv og samfunnssikkerhet:** Skal vi prioritere beskyttelse av privatlivet til enkeltpersoner, eller muligheten til å bruke AI for å øke samfunnets sikkerhet?

### Ta med minst dette

Kunstig intelligens (AI) er dataprogrammer som utfører oppgaver som krever menneskelig intelligens, som mønstergjenkjenning og rask databehandling. AI's utvikling startet på 1950-tallet og har hatt både opp- og nedturer, men har nå fått ny vind i seilene med internett, kraftigere datamaskiner og dyplæring.

AI-verktøy inkluderer maskinlæring (ML) med veiledet læring (SL), ikke-veiledet læring (UL) og forsterket læring (RL). Andre verktøy er naturlig språkprosessering (NLP), datasyn (CV), ansiktsgjenkjenning (FR), kunstige nevrale nettverk (ANN) og dyplæring (DL).

AI brukes i systemutvikling for autofullføring, kodegenerering, feilsøking, testgenerering og dokumentasjonsgenerering, med verktøy som Visual Studio Code, GitHub Copilot og Codeium.

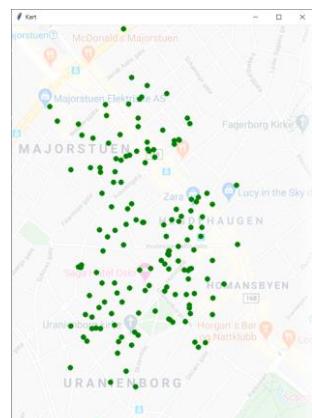
Muligheter med AI finnes i helsevesen, finans, e-handel, produksjon, markedsføring, forsvar og autonome systemer. Utfordringer inkluderer diskriminering, arbeidsledighet, overvåkning, ansvar og sikkerhet. Konsekvenser av AI-bruk er økt automatisering, effektivitet, etiske bekymringer, behov for regulering og økt avhengighet av teknologien.

Etiske dilemmaer innen AI omhandler databeskyttelse, ansvar, ærlighet, diskriminering, kontroll og balansen mellom privatliv og samfunnssikkerhet.

### Øvingsoppgaver

#### 3.10.1

- Se innslaget [Politiet i St Cruz forutsier kriminalitet](#), 5 min.  
(Teknologien som forandrer oss, Første gang sendt: NRK1 torsdag 6. oktober 2016 kl. 19:45)
- Et kartutsnitt av Oslo er vedlagt i filen [kart.png](#). Det er 600x800 piksler. I filen [lokasjoner.csv](#) er det 150 posisjoner (x, y) til simulerte lovbrudd i dette kartet. Lag en app som viser kartet med små sirkler (10 piksler i diameter) i de oppgitte posisjonene.

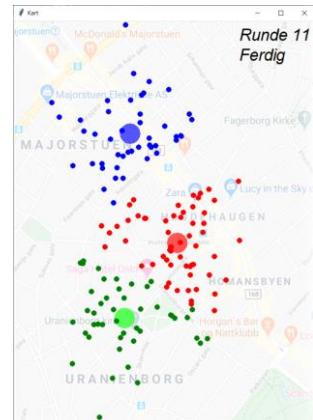


## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

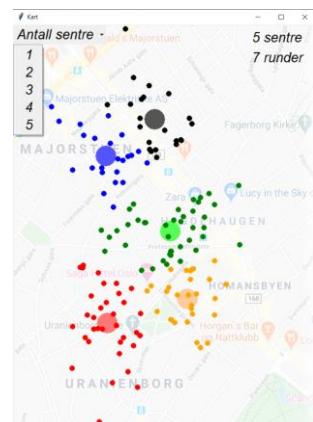
c) Anta at det er stor sannsynlighet for at neste lovbrudd skjer i nærheten av tidligere lovbrudd. Del tidligere lovbrudd inn i tre klynger hvor det står en politibil i midten. Bruk k-grupperings (*k-means*) algoritmen:

- 1) Velg 3 sentre tilfeldig fra de oppgitte lokasjonene og opprett en tom gruppe for hvert senter. Bruk gjerne en liste `grupper` som inneholder lister med punkter for hver gruppe.
- 2) Legg hvert punkt inn i gruppen til det senteret som er nærmest.
- 3) Regn ut gjennomsnittet (x- og y-verdi) i hver gruppe som nye punkter for sentrene.
- 4) Gjenta trinn 2 og 3 til sentrene ikke lenger endrer seg.



Resultatet er 3 klynger av punkter samlet rundt 3 sentre. Tegn større sirkler (radius 20 piksler) i forskjellige farger for sentrene og gi sirklene (radius 5 piksler) for lokasjonene i hver gruppe samme farge som senteret. Legg inn en pause på 2 sekunder mellom hver runde for å se sentrene flytte seg, `canvas.update()` og `canvas.after(2000)`.

- d) Som oppgave c), men nå skal brukeren kunne velge antall sentre, fra og med 1 til og med 5
- e) [Scikit-learn](#) er et populært bibliotek for kunstig intelligens og maskinlæring i Python. Det gir en enkel og brukervennlig API for å ta i bruk en rekke algoritmer, inkludert [\*k-means\*](#) gruppering. `KMeans(5)` returner et objekt av algoritmen med 5 sentre. Dette objektet har en metode `fit` som benytter algoritmen til å tilpasse en modell til datasettet som følger med, `modell = KMeans(5).fit(punkter)`. Lokasjonen til sentrene finner vi i `modell.cluster_centers_`, `modell.labels_` inneholder en indeks for senteret som hvert punkt tilhører og antall runder ligger i `modell.n_iter_`. Gjør oppgave d) på nytt med bruk av [scikit-learn](#) biblioteket.



#### 3.10.2

Drøft det etiske dilemmaet som oppstår når en bedrift vurderer å ta i bruk kunstig intelligens for å automatisere arbeidsprosesser.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 3.11 Oppsummering

*utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger*

Vi har sett at kunstig intelligens kan by på muligheter innen flere bransjer og områder som helsevesenet, finansbransjen, e-handel, forsvar, autonome systemer, markedsføring og produksjon. AI byr også på en rekke utfordringer som diskriminering, arbeidsledighet, overvåkning, ansvar og sikkerhet. Noen konsekvenser ved å ta i bruk AI er automatisering, effektivitet, etiske dilemmaer, regulering og økt avhengighet

*drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn*

Kunstig intelligens fører med seg både fordeler og ulemper. Når vi setter disse opp mot hverandre kan det oppstå etiske dilemmaer, for eksempel mellom databeskyttelse og helse, autonomi og ansvar, ærlighet og kommersielle interesser, diskriminering og effektivitet, kontroll og tillit samt privatliv og samfunnssikkerhet. Vi har drøftet det etiske dilemmaet som oppstår mellom automatisering og økt arbeidsledighet.

*utforske og vurdere alternative løsninger for design og implementering av et program*

**Jupyter** notatbøker egner seg godt til å utforske kode. Vi kan bruke **sett** til å organisere data uten duplikater, og boolske operasjoner som **union**, **snitt** og **differens**, samt boolske operatorer som **and**, **or** og **not**, til å forbedre programdesign og implementering. Vi har sett at **assosiasjon** mellom klasser ofte kan være like viktig som arv for gjenbruk av kode.

**K-means** er en algoritme som brukes i maskinlæring for å gruppere lignende datapunkter i et datasett. Vi har programmert algoritmen selv og etterpå brukt [Scikit-learn](#)-biblioteket for å vurdere alternative løsninger.

*anvende objektorientert modellering til å beskrive et programs struktur*

Klassediagram i objektorientert modellering beskriver et programs struktur. Klassene representerer vanligvis virkelige objekter, og det finnes flere metoder for å finne klassene. Substantivmetoden er en av dem og innebærer å finne alle substantivene i beskrivelsen av problemet og vurdere dem som mulige klasser eller attributter. I UML er det fire forhold mellom klasser: generalisering, assosiasjon, aggregering og komposisjon. Arv er en relasjon som kalles generalisering. Et UML klassediagram viser hvordan klasser, objekter, metoder og arv kan organiseres og visualiseres for å gi en oversikt over programstrukturen

*utvikle objektorienterte programmer med klasser, objekter, metoder og arv*

Vi har fortsatt å utvikle objektorienterte programmer med klasser, objekter, metoder og arv. I programmet med baller viste vi hvordan en Ball klasse kan utvikles, og hvordan objekter av klassen kan opprettes, manipuleres, tegnes og animeres på skjermen. Vi har dessuten utarbeidet en objektorientert mal for innhenting, analyse og presentasjon av informasjon fra reelle datasett

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### vurdere og bruke strategier for feilsøking og testing av programkode

Enhetstesting er en teknikk hvor vi skriver tester for å forsikre oss om at deler av programmet fungerer som planlagt. Tester består av testtilfeller som inkluderer inndata, hvordan testen utføres, og forventet resultat. Flere testtilfeller kan samles i en testsamling som dekker det vi ønsker å teste. Vi har fortsatt å bruke Pytest, som er et populært testverktøy i Python og lar oss velge ut tester og testsamlinger på ulike nivåer. Vi har laget objektorienterte tester som kontrollerer opprettelse av instanser, metoder med returverdier, bivirkninger og unntak. Når alle testene passerer, refaktorerer vi koden for å gjøre den bedre. Til slutt dokumenterte vi resultatet av testene med pytest-html.

### generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

Vi fortsetter å lage gjenbrukbar kode. Når vi oppretter 25 baller i avsnittet om avansert animasjon, gjenbruker alle `ball`-instansene koden fra `Ball`-klassen. Vi har også lært å utvikle gjenbrukbar programkode gjennom assosiasjoner mellom klasser og ved å kalle metoder i andre klasser. I tillegg har vi fortsatt å plassere repeterende kode i egne funksjoner, som `clear_display_frame` i den objektorienterte tkinter App-GUI-malen.

### vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

Når vi lager programmer, er det viktig å tenke på brukervennlighet. Diagrammer kan hjelpe brukeren med å forstå tall bedre, mens animasjoner kan berike brukeropplevelsen. Vi benytter oss av bibliotek som Seaborn for diagrammer, og Pygame for å integrere tekst, grafikk, animasjon og lyd. Ved å kombinere disse verktøyene og elementene på en effektiv måte, kan vi kontinuerlig forbedre brukeropplevelsen i våre og andres programmer.

### velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

Vi har brukt verktøy som *Better Comments*, *autoDocstring* og *pdoc* til å forbedre både koden og dokumentasjonen. Kommentarer i kode og API-dokumentasjon hjelper med å gjøre koden og dokumentasjonen mer forståelig og bidrar til å forbedre samarbeidet mellom utviklere. Dette kan redusere misforståelser og feil når flere mennesker jobber på samme prosjekt.

### gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

I de to første rundene utvekslet vi datafiler med standardformater som `txt`, `csv` og `json`. I denne runden har vi brukt pandas til å håndtere og analysere reelle datasett. Dette verktøyet gjør det mulig å lagre og utveksle data i standardiserte formater som CSV og JSON på en effektiv måte. Vi har også brukt **Jupyter Notebook** for dokumentasjon og formidling. Notatbøker kombinerer kode, tekst og visualiseringer i ett dokument. De lagres i `.ipynb`-formatet, som kan leses av flere programmer som Google Colab og VS Code, og datahåndteringens blir gjennomsiktig og reproducable når den utveksles.

### bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

**pandas** som egner seg godt til innhenting og analyse av data, og `DataFrame`-klassen minner mye om et regneark eller en tabell i en relasjonsdatabase. Vi har lært flere detaljer om pandas og tilnærmet oss med utgangspunkt i kunnskaper om SQL. **Seaborn** er et Python-bibliotek for

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

datavisualisering som gir oss muligheten til å lage flotte diagrammer som er enkelt å kombinere med pandas *dataframes*. **Jupyter** notatbøker kan inneholde formatert (*Markdown*) tekst, kjørbar kode og diagrammer. Sidene kan publiseres til blant annet **html**- og **pdf**-format. Ved hjelp av disse verktøyene kan vi analysere og presentere data fra reelle datasett på en enkel og effektiv måte. Vi har også kombinert pandas med **tkinter** for å lage objektorienterte programmer og utvikle dynamiske løsninger med brukergrensesnitt.

Vurderingsseksemplar

## Runde 4

### 4.1 Brukervennlighet

#### Bolkens innhold

Innledning .....	240
Definisjon.....	241
Kriterier for vurdering .....	241
Grafisk design og universell utforming .....	242
Metoder for vurdering.....	243
Forslag til forbedringer.....	244
Ta med minst dette.....	244
Øvingsoppgaver.....	245

#### Kompetansemål:

- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer

#### Innledning

På den ene siden kan god brukervennlighet føre til fornøyde brukere, økt produktivitet, reduserte opplæringskostnader og større tillit til de som tilbyr systemet. For å oppnå dette kan vi bruke ulike teknikker hvor vi allerede har lært om noen, som for eksempel å formatere tekst, fange opp feil og programmere grafiske brukergrensesnitt, diagrammer, animasjoner og hendelsesstyrte interaksjoner. På den andre siden kan et ikke brukervennlig system føre til frustrasjon, lang tid på å lære seg systemet, redusert produktivitet, økt sannsynlighet for feil og tap av tillit fra brukerne. Eksempelvis har Google, Instagram og Snapchat fått anerkjennelse for sine brukervennlige grensesnitt, mens det har blitt påpekt at noen funksjoner i Facebook og Windows har dårlig brukervennlighet. VIS Visma In School er blitt kritisert for komplekse og uoversiktlige grensesnitt som gjør det vanskelig å finne bestemte funksjoner og innstillinger.



Men hvordan definerer vi egentlig hva som er brukervennlig? Selv om folk flest har erfaring med å bruke nettsider, apper og datasystemer, kan oppfatningen av hva som er brukervennlig være subjektiv og påvirket av personlige preferanser og erfaringer. Derfor er det viktig å

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

gjennomføre objektive tester og involvere et bredt utvalg av brukere i vurderingen av et brukergrensesnitt for å få en representativ og omfattende evaluering.

### Definisjon

*Brukervennlighet kan defineres som en egenskap ved et system som gjør det enkelt å bruke og forstå, uten unødvendige vanskeligheter eller forvirring, slik at brukerne kan utføre sine arbeidsoppgaver effektivt og oppnå tilfredsstillende brukeropplevelse med minst mulig opplæring og feil.*

### Kriterier for vurdering

For å kunne vurdere om et program har god brukervennlighet eller ikke, er det nødvendig å ha klart definerte kriterier. Dette handler ikke bare om hvordan brukergrensesnittet ser ut, men også om faktorer som blant annet funksjonalitet, effektivitet, tilgjengelighet, enkelhet, konsistens, raske og klare tilbakemeldinger, feiltoleranse, sikkerhet og navigasjon. Nedfor vil vi forklare hver av disse faktorene nærmere. Ved å ha disse kriteriene i bakhodet når vi utvikler et program, kan øker vi sjansen for at brukerne blir fornøyde.

- **Funksjonalitet** handler om hvordan programmet fungerer og oppfyller brukernes behov.
- **Effektivitet** vil si at brukerne kan utføre oppgavene sine raskt og enkelt, noe som også innebærer høy oppetid og kort responstid.
- **Tilgjengelighet** handler om å nå flest mulig brukere ved at programmet kan tilpasses brukernes preferanser som eksempelvis skriftstørrelse, kontrast, alternativ navigasjon, talestyring, osv.
- **Enkelhet** handler om hvor lett programmet er å lære, bruke og forstå.
- **Konsistens** vil si at programmet oppfører seg og ser likt ut på tvers av forskjellige funksjoner og deler av programmet.
- **Raske og klare tilbakemeldinger** handler om å slippe å måtte vente, og enkelt kunne forstå de beskjedene system gir.
- **Feiltoleranse** refererer til hvor godt programmet håndterer feil og uforutsette hendelser.
- **Sikkerhet** vil si å beskytte brukernes data og personlige informasjon.
- **Navigasjon** handler om hvor enkelt det er for brukerne å finne veien rundt i programmet til funksjonene de trenger.
- **Grafisk design og universell utforming** er viktig for å skape et brukervennlig grensesnitt, men de er også egne fagområder med en rekke prinsipper som vi omtaler i følgende avsnitt.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Grafisk design og universell utforming

For å skape et brukervennlig grensesnitt er både grafisk design og universell utforming avgjørende. Disse fagområdene er imidlertid så omfattende at de ofte deles opp i flere disipliner eller fagfelt. En av de mest grunnleggende skillene innenfor disse fagområdene er mellom brukeropplevelse (*UX, User Experience*) og brukergrensesnitt (*UI, User Interface*).

Mens en UX-designer kartlegger brukerens vei gjennom systemet, planlegger informasjonsarkitekturen og jobber med *wireframes* og prototyper for å sikre en best mulig brukeropplevelse, fokuserer en UI-designer på det visuelle uttrykket og på å lage gode *mockups*, grafikk og layout ved å velge farger og typografi som bidrar til å skape en enhetlig og sammenhengende design.

Grafisk design er kunsten å bruke visuelle elementer som farger, former, tekst og bilder til å kommunisere informasjon eller ideer til en bestemt målgruppe. Det påvirker hvordan brukerne tolker og forstår informasjonen og kan være med på å skape en brukervennlig opplevelse. Grafisk design handler også om harmoni, balanse, rytme, at noe er felles, likevekt, retning, avstand, intervaller, om linjer, struktur, kontrast, tomrom med mer. Vi bør se grafisk design i sammenheng med de andre kriteriene for vurdering av brukervennlighet og ta hensyn til dette allerede i designfasen. Her en noen generelle råd.

- Er teksten lett å lese, med tilstrekkelig kontrast og tilstrekkelig skriftstørrelse?
- Er informasjonen organisert på en logisk og intuitiv måte?
- Er ikonene og knappene tydelige og lette å forstå, og er de plassert på logiske steder?
- Er fargebruken harmonisk og støtter den opp om navigasjon og bruk av systemet?



**Universell utforming** er en lovpålagt standard som skal sikre at IKT-løsninger kan brukes av så mange mennesker som mulig. Hensikten er å fremme likeverdig samfunnsdeltakelse, hindre digitale barrierer og diskriminering på grunn av nedsatt funksjonsevne. [Forskriften](#) krever blant annet at offentlige virksomheters nettløsninger skal samsvare med kravene i [EN 301 549 V3.2.1](#), som er en europeisk standard på 186 sider. Digitaliseringsdepartementet fører tilsyn etter forskriften, se <https://www.uutilsynet.no/>.



D uutilsynet

Vi skal tilstrebe ett design som møter behovene til ulike brukere ved hjelp av faktorer som kontrast, fargebruk, typografi, plassering av elementer og navigasjon. Dette har fellestrek med

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

tilgjengelighet, men det gir ikke den enkelte bruker muligheten til å innstille brukergrensesnittet etter sine ønsker og behov på samme måte som tilgjengelighet gjør. Et godt grafisk design som tar hensyn til universell utforming vil bidra til å gjøre innholdet enklere å lese og forstå for personer med ulike funksjonsnedsettelser, og samtidig skape en bedre brukeropplevelse for alle brukere.

### Metoder for vurdering

Noen vanlige metoder for å vurdere brukervennlighet er brukertester, ekspertvurderinger, observasjon, analyse av bruksdata og spørreskjemaer

**Brukertester:** Brukere blir bedt om å utføre bestemte oppgaver mens vi observerer og registrerer hvor vellykket de er i å utføre oppgavene. Dette gir verdifull innsikt i hvordan brukerne samhandler med programmet og hvor det eventuelt kan være problemer eller utfordringer. Husk å ha med brukere som er representative for målgruppen for å sikre pålitelige resultater.

**Ekspertvurderinger:** Ekspertes kan for eksempel være designere, utviklere eller andre fagpersoner som har erfaring med å utvikle og evaluere brukervennlige løsninger. Deres vurdering av brukervennligheten kan være et verdifullt tillegg til brukertester, men de kan også være påvirket av personlige preferanser.

**Analyse av bruksdata:** Datalogger kan også gi verdifull informasjon om brukervennligheten. Det kan for eksempel være informasjon om hvor lenge brukerne oppholder seg på ulike sider, eller hvilke funksjoner som blir brukt mest eller minst.

**Spørreskjemaer:** Vi kan lære mye av å be brukerne gi tilbakemeldinger gjennom spørreskjemaer eller tilby en mulighet for å gi tilbakemeldinger eller rapportere problemer direkte i programmet. Det kan fortelle oss hvordan brukerne opplever programmet.

Når vi skal lage et spørreskjema, kan det være lurt å inkludere spørsmål som går detaljert inn på ulike funksjoner og områder av programmet, men det blir spesifikt for hvert enkelt program. Her er noen forslag til mer generelle spørsmål som kan stilles. Bruk en skala fra 1 til 5 eller 1 til 7 der det er mulig, for eksempel 1 = sterkt uenig og 5 = sterkt enig, 1 = veldig vanskelig og 5 = veldig enkelt, 1 = veldig misfornøyd og 5 = veldig fornøyd eller 1 = veldig dårlig og 5 = veldig godt. Ha med en mulighet til å svare "vet ikke" eller "ingen mening" for å utelate disse svarene fra gjennomsnittet på svarskaalen.



1. Programmet er enkelt å bruke.
2. Programmet kan utføre de funksjonene du forventer at det skal gjøre.
3. Er innstillingene og tilpasningsmulighetene i programmet tilstrekkelige for dine behov?
4. Er programmet tilgjengelig på ulike plattformer og enheter?

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

5. Hvor fornøyd er du med søkefunksjonen i programmet?
6. Hvordan vil du beskrive kvaliteten på hjelpefunksjonen og veiledningen i programmet?
7. Er verktøylinjene og menyer i programmet lette å forstå og bruke?
8. Er det lett å navigere i programmet?
9. Hvor fornøyd er du med layout og presentasjon?
10. Hvordan vil du beskrive oppetiden og responstiden til programmet?
11. Har programmet noen feil eller krasjer det ofte?
12. Hvilke funksjoner bruker du mest, og hvorfor?
13. Hvilke funksjoner bruker du minst, og hvorfor?
14. Hva slags funksjoner eller informasjon savner du, og hvorfor?
15. Har du noen generelle tilbakemeldinger eller forslag til forbedringer?

#### **Forslag til forbedringer**

Å forbedre brukergrensesnittet er en kontinuerlig prosess. Brukernes behov og preferanser kan endre seg over tid, og det samme gjelder teknologi. Det som var brukervennlig og relevant i går, kan være utdatert og ineffektivt i dag. Basert på resultatene fra vurderingene, kan vi finne ut hva som bør forbedres. Dette kan være alt fra små endringer i knappeplassering eller layout til nye og bedre funksjoner eller større endringer i navigasjonsstrukturen. Først utforsker vi forskjellige muligheter for å forbedre brukergrensesnittet. Så tester forbedringene på et lite antall brukere før vi lanserer dem. På denne måten kan vi sikre at endringene vi gjør, faktisk fører til en bedre brukeropplevelse.

Til syvende og sist handler brukervennlighet om å utvikle programmer som er enkle å bruke, forstå og tilpasse til individuelle behov. Et brukervennlig grensesnitt kjennetegnes ved at det er lett å lære, effektivt å bruke og gir brukerne tilbakemelding på en klar og tydelig måte. Det bidrar til å øke produktiviteten, redusere frustrasjon, forbedre brukeropplevelsen og bygge tillit. For å klare dette må vi ha god forståelse for brukernes behov og arbeidsmåter, og kontinuerlig vurdere og forbedre brukergrensesnittet basert på klart definerte kriterier. Ved å følge denne framgangsmåten kan vi utvikle programmer som er både funksjonelle og brukervennlige, og som forhåpentligvis tilfredsstiller brukernes behov på best mulig måte.

#### **Ta med minst dette**

Brukervennlighet defineres som hvor enkelt et system er å bruke, og vurderes på bakgrunn av kriterier som funksjonalitet, effektivitet, tilgjengelighet, enkelhet og navigasjon. For å vurdere brukervennlighet, brukes metoder som brukertester, ekspertvurderinger, analyse av bruksdata og spørreskjemaer. Resultatene fra disse vurderingene hjelper oss med å foreslå forbedringer i brukergrensesnittet.

## **Smidig IT-2** **for eksklusiv bruk ved <navn på skole>**

### **Øvingsoppgaver**

Vi skal vurdere brukergrensesnitt til Visma In School og komme med forslag til forbedringer.

**4.1.1**

Intervju 2-3 lærere og 2-3 elever og spør om hvilke funksjoner de bruker i VIS.

**4.1.2**

Lag et spørreskjema basert på svarene i 4.1.1 og forslagene til spørsmål i teksten.

**4.1.3**

Gjennomfør spørreundersøkelsen med minst 5 elever og 5 lærere.

**4.1.4**

Skriv et sammendrag av resultatene.

**4.1.5**

Kom med forslag til å forbedre brukervennligheten i Visma In School

### **TIPS TIL GJENNOMFØRING**

1. I stedet be noen fylle ut et spørreskjema, kan det gå raskere dersom du fyller ut skjemaet for dem i et intervju.
2. Gå sammen i grupper så sammendraget er basert på minst 30 utfylte skjemaer for hver av gruppene elever og lærere.

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 4.2 Hendelser , kollisjoner, tidsstyring og GUI-integrering i Pygame

### Bolkens innhold

Innledning .....	246
Hendelser.....	246
Kollisjoner.....	248
Tidsstyrte handlinger.....	250
Kombinasjon av Tkinter og Pygame.....	251
Ta med minst dette.....	252
Øvingsoppgaver.....	252

### Kompetansemål:

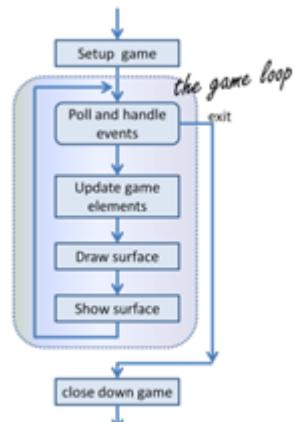
- utforske og vurdere alternative løsninger for design og implementering av et program
- vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv

### Innledning

Da vi utforsket Pygame i bok 3.4, lærte vi å vise tekst, grafikk og animasjoner i et grafisk brukergrensesnitt. Programmene manglet imidlertid muligheten til interaktivitet. Vi kunne ikke påvirke programmene etter at vi hadde startet dem. De levde sitt eget liv. Her skal vi lære hvordan programmene kan respondere på input fra oss mens de kjører. Vi nevnte kort at GUI-programmer er hendelsesstyrte, og nå skal vi forklare nærmere hva dette innebærer.

### Hendelser

Når vi snakker om hendelser i programmering, refererer vi til handlinger eller situasjoner som oppstår mens programmet kjører. Dette kan inkludere brukerens bevegelse av musen, klikk på en knapp, tastetrykk, med mer. Vi vil også utforske tidsstyrte hendelser som inntreffer på spesifikke tidspunkter. I første omgang er det operativsystemet som oppdager disse hendelsene og legger dem i en hendelseskø som blir tilgjengelig for vårt program. Vi har selv kontroll over hvilke hendelser vi ønsker å håndtere, og vi må kontinuerlig sjekke om disse hendelsene har inntruffet før vi oppdaterer og viser noe i vinduet. Dette gjør vi i vår `handle_events()`-metode som kalles fra `run()`-metoden (*The Game Loop*) i Pygame-malen vår. Hvor ofte `update_events()` kjøres, bestemmes av frekvensen som er kontrollert av animasjonshastigheten definert med koden `self.clock.tick(FPS)`, der FPS står for *Frames Per Second*.



# Smidig IT-2

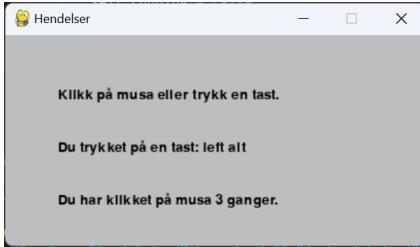
## for eksklusiv bruk ved <navn på skole>

Hendelsehåndtering i Pygame med `pygame.event` gir oss tilgang til flere metoder. I malen bruker vi `pg.event.get()` som returnerer en liste med alle hendelsene som ligger i køen. I [dokumentasjonen](#) (rull ned på siden) og i figuren til høyre kan vi se en liste over ulike hendelsestyper som er implementert. Ruller vi enda lengre ned på siden, finner vi enda flere av dem. Som tidligere nevnt, har vi kontroll over hvilke hendelser vi ønsker å håndtere.

QUIT	none
ACTIVEEVENT	gain, state
KEYDOWN	key, mod, unicode, scancode
KEYUP	key, mod, unicode, scancode
MOUSEMOTION	pos, rel, buttons, touch
MOUSEBUTTONUP	pos, button, touch
MOUSEBUTTONDOWN	pos, button, touch
JOYAXISMOTION	joy (deprecated), instance_id, axis, value
JOYBALLMOTION	joy (deprecated), instance_id, ball, rel
JOYHATMOTION	joy (deprecated), instance_id, hat, value
JOYBUTTONDOWN	joy (deprecated), instance_id, button
JOYBUTTONUP	joy (deprecated), instance_id, button
VIDEORESIZE	size, w, h
VIDEOEXPOSE	none
USEREVENT	code

I vårt første eksempel sjekker vi `KEYDOWN` og `MOUSEBUTTONUP`, i tillegg til standard `QUIT`-hendelsen som allerede er inkludert i malen vår. Vi benytter også to variabler, `self.vis_tastetrykk` og `self.vis_museklikk`, som styrer visningen på skjermen. Disse endres første gang hendelsen oppstår, og lar oss deretter vise innhold i vinduet. I tillegg til dette lagrer vi informasjon om hvilken tast brukeren trykket (`self.tast = pg.key.name(event.key)`) eller antall museklikk (`self.museklikk += 1`). Denne informasjonen vises deretter i `draw`-metoden, med kode vi kjenner igjen fra bolken [3.4](#). Se filen [b\\_4\\_2\\_1\\_event.py](#)

```
def handle_events(self):
    for event in pg.event.get():
        if event.type == pg.QUIT:
            self.running = False
        elif event.type == pg.KEYDOWN:
            self.vis_tastetrykk = True
            self.tast = pg.key.name(event.key)
        elif event.type == pg.MOUSEBUTTONUP:
            self.vis_museklikk = True
            self.museklikk += 1
```



I vårt neste eksempelet tar vi utgangspunkt i filen [b\\_3\\_4\\_6\\_enkel\\_animasjon.py](#) hvor vi animerer en ball. Vi fjerner ballens `update()`-metode og plasserer ballen midt i vinduet ved start.

Deretter endrer vi ballens posisjon med piltastene i `App`-klassens `update_events()`-metode.

Se filen [b\\_4\\_2\\_2\\_flytt\\_ball.py](#).

```
def handle_events(self):
    for event in pg.event.get():
        if event.type == pg.QUIT:
            self.running = False
        elif event.type == pg.KEYDOWN:
            if event.key == pg.K_LEFT:
                self.ball.rect.x -= 10
            elif event.key == pg.K_RIGHT:
                self.ball.rect.x += 10
```



```
def update(self):
    keys = pg.key.get_pressed()
    if keys[pg.K_LEFT]:
        self.rect.x -= 5
    if keys[pg.K_RIGHT]:
        self.rect.x += 5
    self.rect.x = max(0, min(self.rect.x, WIDTH - self.rect.width))
```

Dersom vi ønsker at ballen skal bevege seg kontinuerlig mens vi holder piltasten nede, gjør vi det litt annerledes. Vi må ta i betraktning at i noen tilfeller handler det mer om å håndtere tilstander enn spesifikke hendelser. Derfor håndterer vi ikke denne situasjonen direkte i `App`-klassens `handle_events()`-metode, men sjekker heller hvilke taster som holdes nede i `Ball`-klassens `update()`-metode. Til det bruker vi `pg.key.get_pressed()` for å hente statusen til alle tastene på tastaturet. Disse returneres i en liste vi har kalt `keys`, og vi bruker `keys[pg.K_LEFT]` og `keys[pg.K_RIGHT]` til å sjekke om de tilsvarende piltastene holdes nede. Hvis en av dem er holdt nede, flytter vi ballen i den retningen. Navnene på tilgjengelige taster finner vi i [dokumentasjonen](#) til `key.get_pressed()`. Vi har også lagt til en linje som sørger for at ballen ikke beveger seg utenfor skjermen. Denne tilnærming kan vi bruke når vi skal flytte padden i

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

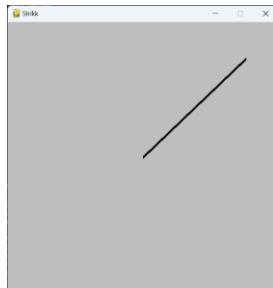
MultiPong oppgaven som vi begynte på i øvingsoppgave [3.4.5](#). Se filen [b\\_4\\_2\\_3\\_styrt\\_animasjon.py](#).

Vi kan også hente ut muspekerens posisjon i vinduet. Det kan vi for eksempel bruke til å tegne en rett linje med `pygame.draw.line` fra midten av lerretet til muspekerens posisjon. For hver oppdatering sletter vi først den forrige linjen (og alt annet i vinduet) så vi oppnår en slags "strikk"-effekt. Vi fanger opp hendelsen `MOUSEMOTION` og henter posisjonen med `event.pos`. Se filen [b\\_4\\_2\\_4\\_strikk.py](#).

```
def handle_events(self):
    for event in pg.event.get():
        if event.type == pg.QUIT:
            self.running = False
        if event.type == pg.MOUSEMOTION:
            self.pos = event.pos

def update(self):
    pass

def draw(self):
    self.screen.fill("grey")
    pg.draw.line(self.screen, 'black', (250, 250), self.pos, width=5)
    pg.display.update()
```



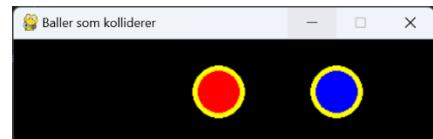
### Kollisjoner

Kollisjoner oppstår når to objekter kommer i kontakt eller overlapper på skjermen. Ofte ønsker vi å reagere på en slik hendelse ved å utføre bestemte handlinger, som å endre objekters retning, registrere treff eller oppdatere poeng. Selv om man kan teste kollisjoner manuelt, slik det er beskrevet i [5.5 Hendelser og animasjon i tkinter](#), gir Pygame oss et sett med funksjoner som forenkler denne prosessen. Ofte involverer kollisjoner *sprites*, og Pygame tilbyr flere metoder for å håndtere slike situasjoner. Det kan være snakk om en *sprite* mot en annen *sprite*, en *sprite* mot flere *sprites* i en *SpriteGroup*, eller ulike *SpriteGroups* mot hverandre.

I eksempelet [b\\_4\\_2\\_5\\_ball\\_mot\\_ball.py](#) har vi to baller som beveger seg mot hverandre. I *App*-klassens `update()`-metode sjekker vi for kollisjon ved hjelp av `pygame.sprite.collide_circle()`. Denne funksjonen returnerer `True` hvis ballenes sirkler overlapper. Når en kollisjon oppdages, kaller vi ballens `collide()`-metode for å håndtere hendelsen, blant annet for å unngå umiddelbar ny kollisjon.

```
def update(self):
    if pg.sprite.collide_circle(self.ball_1, self.ball_2):
        self.ball_1.collide()
        self.ball_2.collide()
    self.all_sprites.update()

def collide(self):
    self.dx *= -1
    self.rect.x += self.dx
```



Det er også andre kollisjonsmetoder som `collide_rect()` og `collide_mask()`. Førstnevnte sjekker om to rektangler overlapper, mens sistnevnte baserer seg på detaljerte masker. `collide_mask()` gir en finjustert deteksjon, som lar oss ha komplekse former som hestesko inne i hverandre uten kollisjon. Den er mer nøyaktig, men kan være tregere enn de andre metodene.

I vårt neste eksempel, [b\\_4\\_2\\_6\\_multipong\\_2.py](#), tar vi utgangspunkt i løsningsforslaget [p\\_3\\_4\\_5\\_multipong\\_1.py](#). Denne gangen introduserer vi padden, som har som oppgave å slå ballene tilbake oppover. For å oppnå dette, har vi laget en egen klasse for padden og deretter opprettet et objekt av denne klassen i *App*-klassens konstruktør. Padden legges også til i spritegruppen `self.all_sprites`.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

I `update()`-metoden sjekker vi kontinuerlig for kollisjoner mellom padden og de andre *sprites* i spritegruppen ved hjelp av `pg.sprite.spritecollide()`. Denne funksjonen sjekker for kollisjoner mellom en *sprite* og en gruppe av *sprites*. Argumentet `False` betyr at vi ikke ønsker å slette *sprites* fra gruppen som kolliderer med vår spesifikke *sprite* (i dette tilfellet padden). Siden padden også er en del av denne gruppen, sjekker vi spesifikt om den kolliderende *spriten* er en ball ved å bruke `isinstance()`-funksjonen.

```
def update(self):  
    # Plasser en ny ball på banen med jevne mellomrom  
    if pg.time.get_ticks() - self.timer >= 1000:  
        self.timer = pg.time.get_ticks()  
        self.all_sprites.add(Ball())  
  
    # Detekter kollisjon mellom padden og ballene  
    hits = pg.sprite.spritecollide(self.padde, self.all_sprites, False)  
    for hit in hits:  
        if isinstance(hit, Ball):  
            hit.speed[1] *= -1  
            # flytt ballen opp for å unngå umiddelbar ny kollisjon  
            hit.rect.top = self.padde.rect.top - hit.rect.height  
  
    self.all_sprites.update()
```

Selv om vi har kollisjonsdeteksjon på plass, har vi enda ikke lagt til funksjonalitet for å bevege padden, eller for å detektere når spillet er over. Dette er igjen som en øvingsoppgave.

I vårt siste eksempel om kollisjoner viser vi hvordan vi kan detektere kollisjoner mellom *sprites* i to forskjellige spritegrupper. Koden i filen `b_4_2_7_spis_ball.py` oppretter fem røde og fem grønne baller. Når de grønne ballene kommer i kontakt med de røde ballene, blir de "spist opp" (slettet).

`groupcollide()`-funksjonen detekterer kollisjoner mellom *sprites* i to grupper og returnerer en ordbok der hver *sprite* i den første gruppen er en nøkkel til en liste av *sprites* fra den andre gruppen den kolliderer med. Denne funksjonen kan også ta hånd om sletting av kolliderende *sprites* direkte, som vi har implementert her. Vi spesifiserer at vi ikke ønsker å slette de røde ballene (`False`), men vi vil slette de grønne ballene (`True`) når en kollisjon detekteres. For kollisjonsdeteksjon bruker vi `collide_circle`, som sjekker om sirkler av to sprites overlapper, i stedet for den vanlige `collide_rect`, som ville sjekket for rektangel-overlapping.

```
def update(self):  
    # Sjekk for kollisjoner og fjern kolliderende grønne baller  
    pg.sprite.groupcollide(  
        self.red_balls,  
        self.green_balls,  
        False,  
        True,  
        collided=pg.sprite.collide_circle,  
    )  
  
    # Oppdater ballposisjoner  
    self.red_balls.update()  
    self.green_balls.update()
```

# Smidig IT-2

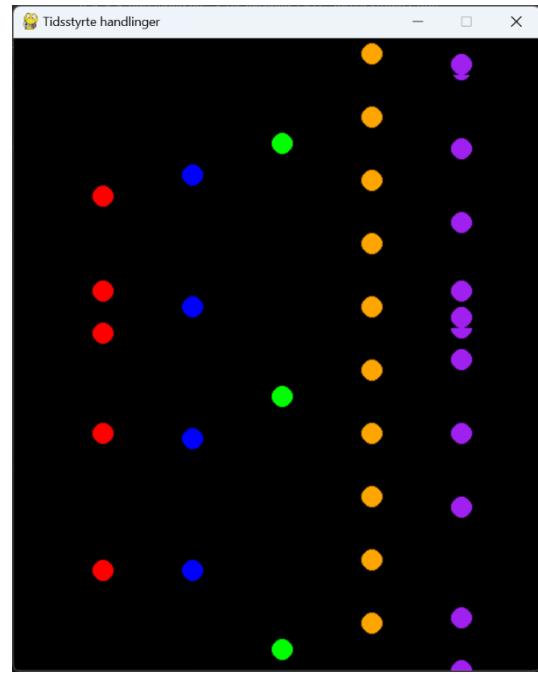
## for eksklusiv bruk ved <navn på skole>

### Tidsstyrte handlinger

Når vi jobber med programmering, er det viktig å kunne kontrollere tiden slik at vi kan utføre oppgaver på bestemte tidspunkter. Dette er nyttig når vi vil automatisere handlinger, som for eksempel å hente data fra nettet på faste tidspunkter. Her er noen måter å gjøre dette på, ved hjelp av koden i filen [b\\_4\\_2\\_8\\_tidsstyring.py](#).

#### Alternativ 1: Tilstfeldige ventetider med `randint()`

Den første måten bruker funksjonen `random.randint()` i vår `update()`-metode. `randint()` gir oss tilfeldige tall som vi kan bruke til å bestemme hvor lenge vi skal vente før vi utfører en handling. Programmet gjentar denne prosessen 24 ganger per sekund på grunn av den definerte bildefrekvensen (FPS). Etter en stund vil tallet 0 dukke opp en gang blant disse tilfeldige tallene, som vi har begrenset til å ligge mellom 0 og 23. Ved å velge tall som er større enn 23 vil ventetiden øke, mens mindre tall vil resultere i kortere ventetid. Derfor blir de røde ballene laget med tilfeldige ventetider, og i gjennomsnitt vil dette skje én gang per sekund.



```
def update(self):
    # ALTERNATIV 1 (Tilstfeldig tidsintervall)
    # legg til en ny rød ball ved tilfeldig tidspunkt
    # i snitt hvert 2. sekund
    if randint(0, FPS - 1) == 0:
        self.all_sprites.add(Ball("red", 100))
```

#### Alternativ 2: Fast ventetid med tidsmåling

Den andre metoden bruker en fast ventetid og måler tiden siden forrige registrerte tidspunkt. I begynnelsen av koden registrerer vi tidspunktet i variablen `self.timer` ved å bruke `pygame.time.get_ticks()`. Denne klokken gir oss antall millisekunder som har gått siden vi startet Pygame med `pg.init()`. I `update()` sjekker vi denne klokken på nytt og utfører handlingen når tilstrekkelig tid har passert. Dette gir oss en pålitelig måte å gjenta handlingen jevnlig. De blå ballene blir opprettet med en fast ventetid på ett sekund.

```
# ALTERNATIV 2 (Fast tidsintervall)
# legg til en ny blå ball hvert sekund
# Husk å sette self.timer = pg.time.get_ticks() i __init__
if pg.time.get_ticks() - self.timer > 1000:
    self.timer = pg.time.get_ticks()
    self.all_sprites.add(Ball("blue", 160))
```

#### Alternativ 3: Egendefinerte hendelser med `set_timer()`

Den tredje måten innebærer å skape spesielle hendelser som inntreffer med jevne mellomrom. Til det bruker vi `pygame.time.set_timer()` og `USEREVENT` i starten av koden. I `handle_events()` sjekker vi om disse hendelsene har skjedd, og hvis de har det, oppretter vi ballene. De grønne ballene blir laget annethvert sekund, mens oransje ballene blir laget to ganger i sekundet.

```
class App:
    def __init__(self):
        pg.init()
        self.clock = pg.time.Clock()
        self.screen = pg.display.set_mode((WIDTH, HEIGHT))
        pg.display.set_caption("pygame mal")
        self.timer = pg.time.get_ticks()
        self.running = True
        self.all_sprites = pg.sprite.Group()
    # ALTERNATIV 3a (Fast tidsintervall)
    # lag en burkerdefinert hendelse hvert 2. sekund
    pg.time.set_timer(pg.USEREVENT, 2000)
    # ALTERNATIV 3b (Fast tidsintervall, men en annen hendelse)
    # lag en annen burkerdefinert hendelse hvert 0.5. sekund
    pg.time.set_timer(pg.USEREVENT + 1, 500)
    # ALTERNATIV 4 (Tilstfeldig tidsintervall, også en annen hendelse)
    # lag en annen burkerdefinert hendelse ved tilfeldig tispunkt
    # i snitt hvert 1. sekund
    pg.time.set_timer(pg.USEREVENT + 2, randint(0, 1000), loops=1)
```

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Alternativ 4: Blande egendefinerte hendelser med tilfeldige ventetider

Den fjerde tilnærmingen kombinerer brukerdefinerte hendelser med tilfeldige ventetider. Ved å bruke `loops=1` lager vi en hendelse som kun utløses én gang. Dette gir oss en enkel og direkte metode for å oppnå både gjennomsnittlige ventetider og fleksibiliteten til å bestemme minimum og maksimum ventetider. Etter å ha opprettet en ball i `handle_events()`, setter vi en ny tilfeldig ventetid ved å kalle `set_timer()` på nytt. Her gir det oss lilla baller med ventetider som varierer mellom 0 og 1 sekund, noe som resulterer i en gjennomsnittlig opprettelseshastighet på omtrent to baller per sekund.

```
def handle_events(self):
    for event in pg.event.get():
        if event.type == pg.QUIT:
            self.running = False
        elif event.type == pg.USEREVENT:
            self.all_sprites.add(Ball("green", 245))
        elif event.type == pg.USEREVENT + 1:
            self.all_sprites.add(Ball("orange", 330))
        elif event.type == pg.USEREVENT + 2:
            self.all_sprites.add(Ball("purple", 415))
    pg.time.set_timer(pg.USEREVENT + 2, randint(0, 1000), loops=1)
```

### Kombinasjon av Tkinter og Pygame

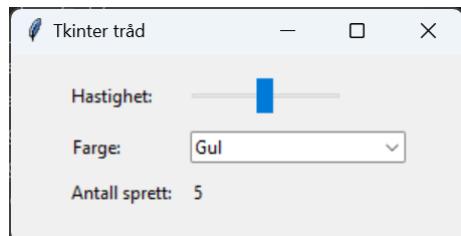
I det første eksempelet, se filen `b_4_2_9_pygame_knapp.py`, benytter vi oss bare av Pygame-biblioteket. Pygame leveres ikke med standard GUI-komponenter som knapper eller tekstfelt. Dette betyr at vi må designe disse fra bunnen av ved hjelp av Pygame's tegnefunksjoner. Selv om dette gir større frihet i GUI-design, kan det også være mer tidkrevende. Som eksempel har vi utarbeidet en `Button`-klasse for å lette opprettelsen av en interaktiv knapp. Med denne klassen kan vi enkelt plassere en knapp hvor som helst i applikasjonen ved å gi den en tekst og posisjon. Den håndterer også rendering og oppdatering av selve knappen. Hvis vi sammenligner med tkinter, ser vi at Pygame ofte krever mer kode for lignende funksjonalitet, og derfor vil vi utforske muligheten for å kombinere Tkinter og Pygame for bedre brukervennlighet.



I det neste eksempelet, se filen `b_4_2_10_pygame_tkinter.py`, har vi en ball som spretter fram og tilbake i et Pygame-vindu. Dette Pygame-vinduet er i sanntid koblet sammen med et Tkinter-vindu ved hjelp av tråder. Tkinter brukes til å lage et GUI som inkluderer en *slider* for hastighetsjustering, en *combobox* for fargevalg og en etikett som viser antall sprett. Bruken av tråder sikrer at begge vinduene kan være åpne og interaktive samtidig. Ved å bruke `queue` for å kommunisere mellom Tkinter og Pygame, kan endringer i GUI-elementene i Tkinter oppdatere animasjonen i Pygame i sanntid. Omvendt kan antall sprett som telles opp i Pygame vises i Tkinter-vinduet ved hjelp av `lock`.



Programmering med tråder anses å være et avansert emne, fordi det krever nøyne synkronisering og kontroll av felles ressurser samtidig som det er vanskelig å feilsøke og rette feil. Tråder er kodeblokker som kjøres samtidig, parallelt, men bare tilsynelatende dersom vi bare har en prosessor med én kjerne. Når deler av programmet kjøres parallelt, kan det oppstå problemer når disse skal ha tilgang til å



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

endre felles ressurser. Vi styrer derfor denne tilgangen. Her har vi gjort det med klassen `SharedResources` som begge trådene har tilgang til, men vi må benytte `queue` eller `lock` for å regulere bruken. `queue` brukes normalt for å sende en melding fra en tråd til en annen, mens `lock` brukes oftere når vi skal ha tilgang til å endre på en variabel.

Vi oppretter en tråd med `threading.Thread()`, starter den med `.start()` og sørger for at hovedtråden venter til tråden er ferdig med `.join()`. Dette sikrer at hovedtråden ikke avslutter før de andre trådene har kjørt ferdig. Vi har også tilpasset koden slik at når vi lukker et av vinduene, lukkes det andre automatisk og appen avsluttes

Å kombinere Tkinter med Pygame kan ses på som en hybrid- eller nød-løsning. Dersom oppgaven krever en mer sofistikert GUI sammen med avanserte animasjoner, kan det være verdt å vurdere andre GUI-biblioteker som PyQt5, Kivy eller wxPython. Imidlertid, med begrenset tid til rådighet og et øye på læreplanens kompetansemål, velger vi å konsentrere oss om Tkinter og Pygame – verktøy vi allerede er kjent med. For å lære mer om Pygame, anbefaler vi nettstedet [Program Arcade Games With Python And Pygame](#).

#### Ta med minst dette

Vi har programmert hendelser, kollisjoner og tidsstyring i Pygame, samt hvordan vi kan kombinere Pygame med Tkinter for å lage mer interaktive og brukervennlige applikasjoner. Vi kan bruke metoden `handle_events` i malen for å sjekke for hendelser som tastetrykk og museklikk. Vi har brukt hendelsestyper som `pg.KEYDOWN`, `pg.KEYUP`, `pg.MOUSEBUTTONDOWN`, `pg.MOUSEBUTTONUP` og `pg.MOUSEMOTION`. For kontinuerlige tastetrykk benytter vi `pg.key.get_pressed()` i en `update`-metode. Objekter vi skal animere, arver fra `Sprite`-klassen for å kunne sjekke for kollisjoner i en `update`-metode ved hjelp av forskjellige `collide`-metoder. Vi kan utføre handlinger på bestemte tidspunkter ved å bruke egendefinerte hendelser med `set_timer()` eller ved å følge med på en klokke eller teller i en `update`-metode. Pygame leveres ikke med standard GUI-komponenter, men vi kan kombinere Pygame med Tkinter ved hjelp av tråder. Dette anses for å være et avansert emne innen programmering.

#### Øvingsoppgaver

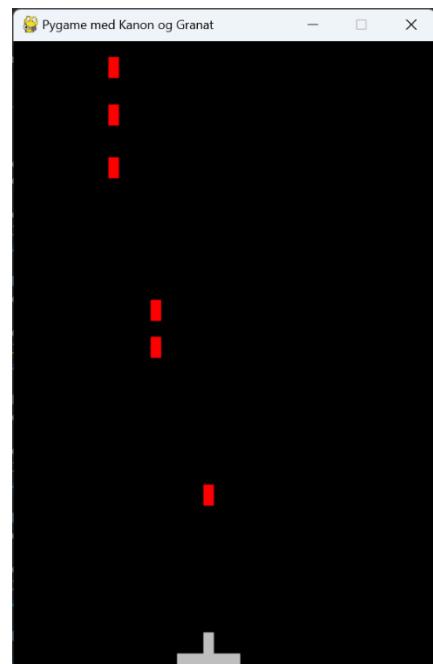
##### 4.2.1

Lag et Pygame program der

- Bredde og høyde er henholdsvis 400 og 600 piksler
- I bunnen skal det være en kanon som vist på figuren. Den skal kunne styres til høyre og venstre med pil tastene, men ikke komme på utsiden av vinduet.
- Når bruker trykker på mellomromstasten, skal kanonen avfyre en granat som kommer ut av munningen og fortsetter oppover på skjermen til den forsvinner og da må den slettes for å spare minne.

##### 4.2.2

Når to baller kolliderer og har samme masse, bytter de hastighet dersom kollisjonen er elastisk. Dette betyr at



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

ingen energi går tapt, og ballene spretter fra hverandre. Endre programmet i filen [b\\_4\\_2\\_5\\_ball\\_mot\\_ball.py](#) så når ballene kolliderer, bytter de hastighet med hverandre.

#### 4.2.3

Ta utgangspunkt i filen [b\\_4\\_2\\_6\\_multipong\\_2.py](#) og fullfør Multipong-oppgaven som er beskrevet i øvingsoppgave [3.4.5](#). Det gjenstår å:

- Implementere bevegelse av padden.
- Avslutte spillet når en ball kommer forbi padden.
- Ekstra utfordring: Når spillet er over, vis i vinduet hvor mange baller spilleren klarte å holde i luften samtidig.

#### 4.2.4

Ta utgangspunkt i filen [b\\_4\\_2\\_7\\_spis\\_ball.py](#) hvor røde baller "spiser" grønne baller. Utvid programmet med følgende slik:

- Når en bruker klikker på et sted i vinduet ([display](#)), opprettes en ny grønn ball på det punktet med en tilfeldig hastighet, lik de andre grønne ballene.
- Når en rød ball "spiser" en grønn ball, skal arealet av den røde ballen øke med arealet til den grønne ballen. Dette betyr at den røde ballens [image \(surface\)](#) og [rect](#) må endres.

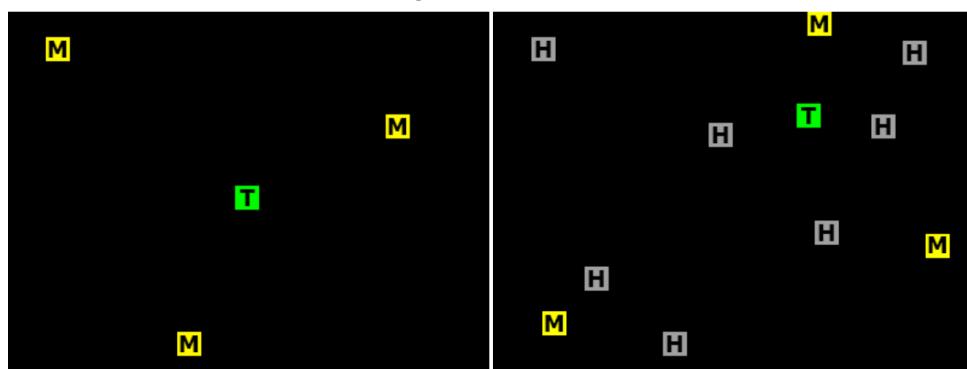
#### 4.2.5

Oppgave 10 – PacTroll fra IT-2 eksamen våren 2023.

I denne oppgaven skal du utvikle et spill som vi har kalt PacTroll. Du bør sette av om lag to timer til denne oppgaven.

Spillet starter med et spillbrett (svart hovedboks), et troll (grønn boks merket T) og tre matbiter (gule bokser merket M) – se illustrasjonen nedenfor til venstre. Trollen skal bevege seg på spillbrettet og spise så mange matbiter som mulig. Trollen har en konstant fart og kan ikke stoppes, men spilleren styrer retningen ved å bruke tastaturet. Hver gang en matbit spises, blir den gjort om til en hindring (grå boks merket H), og en ny matbit plasseres ut på spillbrettet – se illustrasjonen nedenfor til høyre.

Hvis trollen treffer en av hindringene eller spillbrettets kanter, avsluttes spillet.



## **Smidig IT-2** **for eksklusiv bruk ved <navn på skole>**

Funksjonelle krav:

- Ved oppstart skal grensesnittet se ut omtrent som i illustrasjonen ovenfor til venstre. Spillet skal bestå av et spillbrett, et trolleyobjekt og tre matobjekter. (Du kan utelate bokstavene på boksene.)
- Ved oppstart plasseres matobjektene på tre tilfeldige plasseringer på spillbrettet, og trolleyobjektet plasseres i sentrum. Ingen objekter skal være oppå hverandre.
- Trolleyobjektet beveger seg i rolig hastighet i en retning, enten opp, ned, til venstre eller til høyre. Retningen trolleyobjektet beveger seg i, styres med pil-taster (alternativt tastene W, S, A og D).
- Når trolleyobjektet treffer et matobjekt, skal det følgende skje:
  - Spilleren får et poeng.
  - Matobjektet gjøres om til et hindringsobjekt som trolleyobjektet ikke skal treffe igjen.
  - Et nytt matobjekt opprettes et tilfeldig sted på spillbrettet, slik at det alltid er tre matobjekter på spillbrettet. Nye matobjekter skal ikke plasseres på allerede eksisterende objekter.
  - Farten til trolleyobjektet økes.
- Antallet poeng spilleren har, skal hele tiden være synlig i grensesnittet.
- Spillet avsluttes om trolleyobjektet treffer en av spillbrettets kanter eller et av hindringsobjektene.

Oppdrag:

- a. Lag et klassediagram der du definerer nødvendige klasser, med egenskaper og metoder, for å implementere/lage spillet.
- b. Implementer/lag spillet i samsvar med klassediagrammet fra A.

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 4.3 Datarensing og datautforskning

### Bolkens innhold

Innledning .....	255
Datarensing.....	256
Bli kjent med dataene .....	257
Rette opp feilaktige data .....	259
Fjerne duplikater.....	259
Håndtere ulike formater.....	260
Håndtere manglende verdier.....	262
Datautforskning .....	262
Enkle sentral- og spredningsmål .....	262
Sorteringer .....	263
Grupperinger.....	263
Diagrammer.....	264
Ta med minst dette.....	265
Øvingsoppgaver.....	265

### Kompetanse mål:

- bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett
- utforske og vurdere alternative løsninger for design og implementering av et program
- vurdere og bruke strategier for feilsøking og testing av programkode
- gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

### Innledning

Ofte er datasettene vi jobber med i IT-2 allerede renset, men det er ikke alltid tilfellet, slik vi opplevde det under eksamen våren 2023. Et typisk datavitenskapsprosjekt består av flere faser for å finne løsninger eller svare på spørsmål vi har. For eksempel:

- **Problemdefinisjon:** identifisere og formulere klare og konkrete problemer eller mål for prosjektet.
- **Datainnsamling:** Hente relevante data fra pålitelige kilder.
- **Datarensing:** Rense og forberede dataene ved å behandle ulike formater, rette opp feilaktige data, håndtere manglende verdier og fjerne duplikater, blant andre oppgaver.
- **Datautforskning:** Utforske og analysere dataene for å forstå egenskaper, sammenhenger, mønstre og avvik.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

- **Dataanalyse:** Bruke statistiske metoder, modeller og algoritmer for å finne svar på problemet eller målet.
- **Dataproduct:** Presentere resultatet på en forståelig måte basert på analysen av datasettet.

De to første fasene er ikke vektlagt i kompetansemålet ovenfor. Vi skal vi hente inn informasjon fra reelle datasett, men ikke samle inn data. Dette innebærer vanligvis å lese inn data fra en lokal fil eller foreta en URL-forespørsel på Internett. Vi kan utforske dataene, men har begrensninger innen dataanalyse og dataproduct ettersom mye av statistikken er fjernet fra de nye læreplanene (LK20). Imidlertid inkluderer datautforskingen både analyse og presentasjon på et mer grunnleggende nivå, så på mange måter kan vi si at vi slår sammen de tre siste fasene til én fase med hovedfokus på datautforskning.

### **Datarensing**

Datarensing handler om å klargjøre dataene for videre utforskning og analyse, og inkluderer aktiviteter som å gjøre seg kjent med dataene, behandle ulike formater, rette opp feilaktige data, håndtere manglende verdier og fjerne duplikater.

Videre i teksten skal vi jobbe med datasettet som ble gitt til IT-2 eksamen våren 2023:

### **"Datasett med apper i Google Play Store**

Google Play Store er en appbutikk som tilbyr kjøp og nedlasting av apper for Android-telefoner. Vedlagt ligger et datasett som inneholder statistikk over apper i Google Play Store, i CSV- og JSON-format. Du kan velge hvilket av formatene du vil bruke. Datasettet kan lastes ned fra følgende lenker:

- [CSV](#)
- [JSON](#)

Last ned datasettet i ønsket format og gjør deg kjent med det. Du må også forberede deg på å behandle datasettet med programmering og tilpasse data i settet for å kunne gjøre aktuelle beregninger. I tillegg må du kontrollere at du har de nødvendige ressursene installert på den datamaskinen du skal bruke på eksamenen. På eksamenen vil du få to oppgaver hvor du skal behandle dette datasettet og presentere informasjon fra det med programmering.

Datasettet er kodet med tegnsettet UTF-8. Det kan være nødvendig å angi tegnsett når du skal lese datasettet med programmering.

Datasettet har følgende nøkler:

- App, Category: tekst
- Rating: desimaltall
- Reviews: heltall
- Size: heltall og bokstav (antall kB eller MB)
- Installs: tekst som inneholder heltall og + (størrelsesorden)
- Type: boolsk ("paid" eller "free")
- Price: heltall

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

- Content Rating, Genres: tekst
- Last Updated: dato
- Current Ver, Android Ver: tekst

Separatoren i CSV-filen er komma."

### Bli kjent med dataene

Vi åpner CSV-filen i Excel for å bli kjent med dataene, og det kan være nyttig å bla eller scrollere litt rundt for å få en følelse av hvilke verdier som ligger i cellene.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159 19M	10,000+	Free	0 Everyone	0	Art & Design		07.01.2018	1.0.0	4.0.3 and up
3	Coloring book moana	ART_AND_DESIGN	3.9	967 14M	500,000+	Free	0 Everyone	0	Everyone	Art & Design;Pretend Play	15.01.2018	2.0.0	4.0.3 and up
4	UI Launcher Lite –FREE Live Cool Themes, Hide Apps	ART_AND_DESIGN	4.7	87510 8.7M	5,000,000+	Free	0 Everyone	0	Everyone	Art & Design	01.08.2018	1.2.4	4.0.3 and up

Det er 13 kolonner som stemmer med de oppgitte nøklene (kodeboka). Trykker vi Ctrl+↓, kommer vi til siste utfylte rad, som er rad nummer 10842. Dersom vi trekker fra overskriftsraden, sitter vi igjen med 10841 rader med data eller observasjoner. De fleste kolonnene har tekstverdier, fordi innholdet er venstrejustert. Eksempelvis er *Rating* sannsynligvis tekst for det er brukt punktum i stedet for komma. Slik kan vi rette opp i senere når vi importerer dataene til Python og bruker pandas.

Når vi sorterer fra A til Å på *App*, ser vi i rad 21 og 22 at datasettet inneholder duplikater.

	A	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
21	10 Best Foods for You	HEALTH_AND_FITNESS	4.0	2490 3.8M	500,000+	Free	0 Everyone	0	Everyone 10+	Health & Fitness	17.02.2017	1.9	2.3.3 and up
22	10 Best Foods for You	HEALTH_AND_FITNESS	4.0	2490 3.8M	500,000+	Free	0 Everyone	0	Everyone 10+	Health & Fitness	17.02.2017	1.9	2.3.3 and up

Når vi sorterer fra Å til A på *Rating*, ser vi at flere (1474) av radene inneholder NaN (*Not a Number*)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2	"1 DT" Fútbol. Todos Somos Técnicos.	SPORTS	NaN	27.3M	500+	Free	0 Everyone	0	Everyone	Sports	07.10.2017	0.22	4.1 and up
3	[EF]ShoutBox	COMMUNICATION	NaN	3.25M	100+	Free	0 Teen	0	Teen	Communication	07.08.2018	Build 54	4.0.3 and up
4	14thStreetVet	MEDICAL	NaN	0.29M	5+	Free	0 Everyone	0	Everyone	Medical	16.07.2018	3000000.11	4.0.3 and up

*Reviews* ser grei ut, men når vi sorterer fra størst til minst, blir vi igjen minnet på at datasettet inneholder duplikater. Denne gangen er det også marginale forskjeller i antall *Reviews*.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2	Facebook	SOCIAL	4.1	7815830K	Varies with device	1,000,000+	Free	0	Teen	Social	03.08.2018	Varies with device	Varies with device
3	Facebook	SOCIAL	4.1	7812820K	Varies with device	1,000,000+	Free	0	Teen	Social	03.08.2018	Varies with device	Varies with device
4	WhatsApp Messenger	COMMUNICATION	4.4	69119316K	Varies with device	1,000,000+	Free	0	Everyone	Communication	03.08.2018	Varies with device	Varies with device
5	WhatsApp Messenger	COMMUNICATION	4.4	69119316K	Varies with device	1,000,000+	Free	0	Everyone	Communication	03.08.2018	Varies with device	Varies with device
6	WhatsApp Messenger	COMMUNICATION	4.4	69109672K	Varies with device	1,000,000+	Free	0	Everyone	Communication	03.08.2018	Varies with device	Varies with device
7	Instagram	SOCIAL	4.5	66577446K	Varies with device	1,000,000+	Free	0	Teen	Social	31.07.2018	Varies with device	Varies with device
8	Instagram	SOCIAL	4.5	66577313K	Varies with device	1,000,000+	Free	0	Teen	Social	31.07.2018	Varies with device	Varies with device
9	Instagram	SOCIAL	4.5	66577313K	Varies with device	1,000,000+	Free	0	Teen	Social	31.07.2018	Varies with device	Varies with device
10	Instagram	SOCIAL	4.5	66509917K	Varies with device	1,000,000+	Free	0	Teen	Social	31.07.2018	Varies with device	Varies with device

Når vi sorterer fra Å til A på *Size*, ser vi at flere (1695) av radene inneholder teksten "Varies with device".

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2	20 Minuten (CH)	NEWS_AND_MAGAZINE	3.5	14153K	Varies with device	1,000,000+	Free	0	Everyone 10+	News & Magazines	03.08.2018	Varies with device	Varies with device
3	20 minutes (CH)	NEWS_AND_MAGAZINE	3.7	4379K	Varies with device	1,000,000+	Free	0	Teen	News & Magazines	03.08.2018	Varies with device	Varies with device
4	2018Emoji Keyboard Emoticons Lite -sticker&gif	PERSONALIZATION	4.2	115773K	Varies with device	10,000,000+	Free	0	Everyone	Personalization	22.05.2018	Varies with device	4.1 and up

Når vi sorterer fra A til Å på *Installs*, ser vi at en rad inneholder bare 0, ikke etterfulgt av +.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2	Command & Conquer: Rivals	FAMILY	NaN	0	Varies with device	0	NaN	0	Everyone 10+	Strategy	28.06.2018	Varies with device	Varies with device
3	AK Parti Yardım Toplama	SOCIAL	NaN	0	8.7M	0+	Paid	Teen	0	Social	28.07.2017	3.4.4.3.3	4.1 and up
4	AP Series Solution Pro	FAMILY	NaN	0	7.4M	0+	Paid	Everyone	0	Education	30.07.2017	1.3	4.0 and up

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi sorterer fra Å til A, ser vi raden som det ble så mye ståhei om under eksamen, hvor verdiene virker forskjøvet. Ut fra verdiene i denne raden ser det ikke ut som det vil påvirke analysen og presentasjonen dersom denne raden droppes.

A	B	C	D	E	F	G	H	I	J	K	L	M
App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
2 Life Made Wi-Fi Touchscreen Photo Frame	1.9	19	1,000+		Free	0			February 11, 2018	4.0 and up		
3 EvAk Smart Home	LIFESTYLE	NaN	14	Varies with device	500+	Free	0	Everyone	Lifestyle	13.03.2018 1.1.1	4.1 and up	
4 MediBeat for AW - Android (1)	HEALTH_AND_FITNESS	NaN	2	Varies with device	500+	Free	0	Everyone	Health & Fitness	24.10.2017 0.9.16	Varies with device	

Type ser grei ut. De fleste er Free, bare 800 er Paid, og det finnes én 0 og én NaN. Variabelen er oppgitt til å være boolsk, men den er strengt talt tekst.

Det ser ut som Excel ikke har klart å lese inn Price (Paid) når den ikke er 0 (Free). Igjen skyldes dette sannsynligvis de regionale innstillingene, og vi vil ordne dette opp i Python. Hvis vi åpner CSV-filen i en teksteditor, ser vi at Price er for eksempel oppgitt som \$4.99, som er et desimaltall, og ikke et heltall som angitt.

De resterende kolonnene ser greie ut. Current Ver og Android Ver har mellom 1300 og 1500 Varies with device og noen få NaN. I Last Updated er datoene oppgitt innenfor doble anførselstegn.

For å oppsummere, etter en første gjennomgang av dataene ser vi at flere kolonner mangler over 10% av verdiene. Det er også en betydelig mengde duplikater, og flere verdier må konverteres. I tillegg er det minst én feilaktig rad i datasettet. Vi vil ta tak i disse punktene og håndtere dem videre i behandlingen av datasettet ved hjelp av programering.

Vi er nå klare til å lese inn filen til en pandas.DataFrame. Først ser vi hvordan de første radene ser ut, head(), og ber deretter om informasjon om datasettet info().

```
import pandas as pd
import locale
locale.setlocale(locale.LC_ALL, 'nb_NO.UTF-8')

df = pd.read_csv('googleplaystore.csv')
display(df.head())
display(df.info())
for item in df.iloc[0]:
    print(type(item))
    ✓ 0.1s
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0	4.0.3 and up
1	Coloring book moana	ART_AND DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

Som forventet har vi lest inn 13 kolonner og 10841 rader. Kun Rating har blitt lest inn som desimaltall, resten er tekstverdier. For å undersøke hva som gjemmer seg bak object i Dtype-kolonnen, kan vi skrive ut datatypen til verdiene i den første raden:

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   App          10841 non-null   object  
 1   Category     10841 non-null   object  
 2   Rating       9367 non-null   float64 
 3   Reviews      10841 non-null   object  
 4   Size          10841 non-null   object  
 5   Installs     10841 non-null   object  
 6   Type          10840 non-null   object  
 7   Price         10841 non-null   object  
 8   Content Rating 10840 non-null   object  
 9   Genres        10841 non-null   object  
 10  Last Updated 10841 non-null   object  
 11  Current Ver 10833 non-null   object  
 12  Android Ver 10838 non-null   object  
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
None
```

### Rette opp feilaktige data

#### Rette opp i feilaktige data

For å unngå potensielle problemer senere, fjerner vi raden med feil umiddelbart.

```
display(df[df['App'] == 'Life Made WI-Fi Touchscreen Photo Frame'])
df.drop(10472, inplace=True)
print(f'Antall rader: {len(df)}')
✓ 0.1s
```

	App	Category	Rating	Reviews	Size	Installs
10472	Life Made WI-Fi Touchscreen Photo Frame		1.9	19.0	3.0M	1,000+

Antall rader: 10840

### Fjerne duplikater.

#### Fjerne duplikater

Først sjekker vi hvor mang duplikater det er i App-kolonnen  
Når en App har flere rader, velger vi å beholde den med flest anmeldelser (Reviews)  
Da må vi konvertere Reviews til int.

```
print(f'Antall App-duplikater: {df.duplicated(subset='App').sum()}')
df['Reviews'] = pd.to_numeric(df['Reviews'], errors='coerce')
print(f'Antall NaN i Reviews: {df["Reviews"].isna().sum()}')
df = df.sort_values(by='Reviews', ascending=False)
df = df.drop_duplicates(subset='App', keep='first')
print(f'Antall rader igjen: {len(df)}')
display(df.head(3))
✓ 0.0s
```

Antall App-duplikater: 1184  
Antall NaN i Reviews: 0  
Antall rader igjen: 9656

	App	Category	Rating	Reviews	Size	Installs
2544	Facebook	SOCIAL	4.1	78158306	Varies with device	1,000,000,000+
381	WhatsApp Messenger	COMMUNICATION	4.4	69119316	Varies with device	1,000,000,000+
2604	Instagram	SOCIAL	4.5	66577446	Varies with device	1,000,000,000+

errors='coerce' vil si at ugyldige verdier blir satt til NaN.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Håndtere ulike formater

Category er tekst. Vi kan sjekke om noen har skrivefeil ved å sjekke unike kategorier. Nedenfor viser vi bare de tre første av 33 kategorier.

### Håndtere ulike formater

```
print(f'Antall unike kategorier: {len(df["Category"].unique())}')
for cat in df['Category'].unique():
    print(cat)
✓ 0.0s
Antall unike kategorier: 33
SOCIAL
COMMUNICATION
GAME
```

Rating er lest inn som desimaltall, men vi kan jo sjekke antall NaN.

```
print(f'Rating er av type {type(df["Rating"].iloc[0])}')
print(f'Antall NaN i Rating: {df["Rating"].isna().sum()}')
print(df["Rating"].describe())
✓ 0.1s
Rating er av type <class 'numpy.float64'>
Antall NaN i Rating: 1463
count    8193.000000
mean      4.173245
std       0.536345
min      1.000000
25%     4.000000
50%     4.300000
75%     4.500000
max      5.000000
Name: Rating, dtype: float64
```

Når vi konverterer M til millioner og k til tusen i Size-kolonnen, oppstår det mange NaN-verdier for de appene som har teksten *Varies with device* i stedet for numeriske verdier. Dette gjelder blant annet populære apper som *Facebook*, *WhatsApp Messenger* og *Instagram*. Det er viktig å være oppmerksom på disse NaN-verdiene når vi senere skal analysere og presentere informasjonen. Her har vi opprettet en ny kolonne, **Størrelse**, for å lettere kontrollere at konverteringen går riktig for seg.

```
def convert_size(size):
    try:
        if size[-1] == 'M':
            return float(size[:-1]) * 1_000_000
        elif size[-1] == 'k':
            return float(size[:-1]) * 1_000
        elif size.isnumeric():
            return float(size)
        else:
            return math.nan
    except (ValueError, IndexError):
        return math.nan

df['Størrelse'] = df['Size'].apply(convert_size)
print(f'Antall NaN i Størrelse: {df["Størrelse"].isna().sum()}')
display(df[['App', 'Size', 'Størrelse']].sample(n=10).head(10))
✓ 0.0s
Antall NaN i Størrelse: 1228
```

	App	Size	Størrelse
2677	EHS Dongsen Shopping	9.0M	9000000.0
8007	Free CW TN Videos 2018	4.8M	4800000.0
8386	DG Users	3.1M	3100000.0
1225	Easy Healthy Recipes	7.2M	7200000.0
1316	Daily Workouts - Exercise Fitness Routine Trainer	Varies with device	NaN
5400	Resume Builder Free, 5 Minute CV Maker & Templ...	6.7M	6700000.0
5033	ActionDirector Video Editor - Edit Videos Fast	32M	32000000.0
3408	Kairo XP (for HD Widgets)	779k	779000.0
3375	ZenUI Themes – Stylish Themes	7.2M	7200000.0
5632	Nights at Cube Pizzeria 3D – 3	40M	40000000.0

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

**Installs**-kolonnen byr på noen utfordringer. Først må vi fjerne kommaene. Deretter sjekker vi om verdien er '0', og hvis den er det, setter vi den til 0. Hvis verdien slutter med '+', fjerner vi '+'-tegnet og konverterer resten til et heltall. Hvis verdien allerede er et heltall, konverterer vi den direkte. Den statistiske utfordringen ligger i hvilken verdi vi skal bruke innenfor intervallene, men vi bruker nedre grense for enkelhetens skyld. Vi har opprettet en ny kolonne, **Installasjoner**, for å lettere kontrollere konverteringen.

**Price** er lett å konvertere når vi fjerner \$-tegnet, og vi ser at tallene stemmer med **Type**-kolonnen.

```
def convert_installs(installs):
    try:
        installs = installs.replace(',', '')
        if installs == '0':
            return 0
        elif installs[-1] == '+':
            return int(installs[:-1])
        else:
            return int(installs)
    except (ValueError, IndexError):
        return math.nan

df['Installasjoner'] = df['Installs'].apply(convert_installs)
print(f'Antall NaN i Installs: {df["Installs"].isna().sum()}')
display(df[['App', 'Installs', 'Installasjoner']].sample(n=5))
✓ 0.0s

Antall NaN i Installs: 0

          App      Installs  Installasjoner
6543  Bangla Calendar 1425: (EN-BN-AR) Holiday   1,000+       1000
3492                  myAT&T  50,000,000+  50000000
9890                 TAXLANDIA   1,000+       1000
3295  Flashlight - Torch LED Light  10,000,000+ 10000000
7123      Minwa HA (C.B.)      50+         50
```

```
df['Price'] = df['Price'].str.replace('$', '')
df['Price'] = pd.to_numeric(df['Price'], errors='coerce')
print(f'Antall Paid: {df[df["Type"] == "Paid"].shape[0]}')
print(f'Antall pris > 0: {df[df["Price"] > 0].shape[0]}')
print(f'Antall Free: {df[df["Type"] == "Free"].shape[0]}')
print(f'Antall pris == 0: {df[df["Price"] == 0].shape[0]}')
print(f'Antall NaN i Price: {df["Price"].isna().sum()}')
print(f'Antall NaN i Type: {df["Type"].isna().sum()}')
✓ 0.1s

Antall Paid: 754
Antall pris > 0: 754
Antall Free: 8901
Antall pris == 0: 8902
Antall NaN i Price: 0
Antall NaN i Type: 1
```

De resterende kolonnene kan inntil videre stå som de er, bortsett fra **Last Updated** som vi bruker til å lage en ny kolonne, **Sist oppdatert**, med pandas **datetime**-objekter.

```
df['Sist oppdatert'] = pd.to_datetime(df['Last Updated'], format='%B %d, %Y')
display(df[['App', 'Last Updated', 'Sist oppdatert']].sample(n=5))
✓ 0.1s

          App      Last Updated  Sist oppdatert
421  Ghostery Privacy Browser  March 20, 2018  2018-03-20
4817           War Z 2        April 3, 2018  2018-04-03
9630  Orfox: Tor Browser for Android  July 2, 2018  2018-07-02
7701  Evolve CP Calc. for PokemonGo December 18, 2016  2016-12-18
1646 Bed Time Fan - White Noise Sleep Sounds  April 26, 2018  2018-04-26
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Håndtere manglende verdier

Vi har nå oversikt over antallet manglende verdier i hver av kolonnene. For øyeblikket trenger vi ikke å gjøre ytterligere tiltak med dataene, men når vi skal analysere informasjonen, må vi være spesielt oppmerksomme på kolonnene **Størrelse** og **Rating**, som har manglende data i over 10% av radene. Dette gjelder spesielt **Størrelse**, ettersom den mangler for flere av appene med flest installasjoner, som for eksempel *Facebook*, *WhatsApp Messenger* og *Instagram*. Vi bør også være klar over at både **Current Ver** og **Android Ver** inneholder over 1000 rader med *Varies with device*, men dette forventes ikke å utgjøre noen problemer.

```
Manglende verdier

print(df.isna().sum())
print(df.info())
✓ 0.1s

App           0
Category       0
Rating        1463
Reviews        0
Size           0
Installs       0
Type           1
Price           0
Content Rating 0
Genres          0
Last Updated    0
Current Ver      8
Android Ver      2
Størrelse      1228
Installasjoner  0
Sist oppdatert   0
dtype: int64
```

### Datautforsking

Datautforsking handler, som nevnt tidligere, om å utforske og analysere dataene for å få innsikt i egenskaper, sammenhenger, mønstre og avvik. Det er rimelig å anta at det kan være en sammenheng mellom *Rating* og *Sist oppdatert*. Apper som oppdateres hyppig, vil kanskje være nyligere oppdatert og kan muligens få en høyere *Rating*. Det samme gjelder for *Installasjoner* og *Reviews*. Apper med flere installasjoner får sannsynligvis flere anmeldelser. Apper med større filstørrelse kan også ha mer funksjonalitet og bedre brukervennlighet, noe som kan påvirke antall *Installasjoner* og *Rating* positivt. Denne typen utforskning overskridet kravene vi mener er nødvendige i IT-2. Vi vil begrense oss til å identifisere enkle egenskaper som minimum, gjennomsnitt og maksimum, samt utføre ulike former for sortering, gruppering og presentasjon av dataene gjennom tabeller og diagrammer.

```
<class 'pandas.core.frame.DataFrame'>
Index: 9656 entries, 2544 to 5086
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   App               9656 non-null   object  
 1   Category          9656 non-null   object  
 2   Rating             8193 non-null   float64 
 3   Reviews            9656 non-null   int64   
 4   Size               9656 non-null   object  
 5   Installs           9656 non-null   object  
 6   Type               9656 non-null   object  
 7   Price              9656 non-null   float64 
 8   Content Rating    9656 non-null   object  
 9   Genres             9656 non-null   object  
 10  Last Updated      9656 non-null   object  
 11  Current Ver       9648 non-null   object  
 12  Android Ver       9654 non-null   object  
 13  Størrelse          8428 non-null   float64 
 14  Installasjoner     9656 non-null   int64   
 15  Sist oppdatert     9656 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(3), int64(2), object(10)
memory usage: 1.3+ MB
None
```

### Enkle sentral- og spredningsmål

- *Rating*: Gjennomsnittlig vurdering av appene er rundt 4.2, med en minimumsvurdering på 1 og en maksimumsvurdering på 5.
- *Reviews*: Gjennomsnittlig antall anmeldelser er rundt 216 870, med minimum 0 anmeldelser og maksimum 78 158 310 anmeldelser.

Datautforskning

### Enkle sentral- og spredningsmål

```
display(df[['Rating','Reviews','Price','Størrelse',
           'Installasjoner','Sist oppdatert']].describe())
✓ 0.1s
```

	Rating	Reviews	Price	Størrelse	Installasjoner	Sist oppdatert
count	8193.000000	9.656000e+03	9656.000000	8.428000e+03	9.656000e+03	9656
mean	4.173245	2.168709e+05	1.097571	2.040191e+07	7.800478e+06	2017-10-30 23:48:13.123446784
min	1.000000	0.000000e+00	0.000000	8.500000e+03	0.000000e+00	2010-05-21 00:00:00
25%	4.000000	2.500000e+01	0.000000	4.575000e+06	1.000000e+03	2017-08-07 00:00:00
50%	4.300000	9.690000e+02	0.000000	1.200000e+07	1.000000e+05	2018-05-04 12:00:00
75%	4.500000	2.946775e+04	0.000000	2.800000e+07	1.000000e+06	2018-07-17 00:00:00
max	5.000000	7.815831e+07	400.000000	1.000000e+08	1.000000e+09	2018-08-08 00:00:00

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

- *Price*: Gjennomsnittlig pris er omrent 1.1, med ingen pris for enkelte apper og en maksimal pris på 400.
- *Størrelse*: Gjennomsnittlig størrelse på appene er rundt 20.4 MB med en minimumsstørrelse på 8.5 kB og en maksimal størrelse på 100 MB.
- *Installasjoner*: Gjennomsnittlig antall installasjoner er rundt 7.8 millioner, med ingen installasjoner for noen apper og en maksimal installasjon på 1 milliard.
- *Sist oppdatert*: Varierer fra 2010 til 2018.

### Sorteringer

#### Sorteringer

```
print('Apper sortert etter Rating')
display(df.sort_values(by='Rating', ascending=False) \
[[['App', 'Category', 'Rating']] \
.head(3).set_index('App'))
```

✓ 0.1s

Apper sortert etter Rating

App	Category	Rating
Helping BD	LIFESTYLE	5.0
Flippy Axe : Flip The Knife & Axe Simulator	GAME	5.0
COMSATS BOOK STORE FOR BS(CS)	FAMILY	5.0

```
print('Apper sortert etter Reviews')
display(df.sort_values(by='Reviews', ascending=False) \
[[['App', 'Category', 'Reviews']] \
.head(3).set_index('App'))
```

✓ 0.0s

Apper sortert etter Reviews

App	Category	Reviews
Facebook	SOCIAL	78158306
WhatsApp Messenger	COMMUNICATION	69119316
Instagram	SOCIAL	66577446

```
print('Apper sortert etter flest Installasjoner og dårligst Rating')
display(df.sort_values(by=['Installasjoner', 'Rating'], \
ascending=[False, True]) \
[[['App', 'Category', 'Installasjoner', 'Rating']] \
.head(5).set_index('App'))
```

✓ 0.5s

Apper sortert etter flest Installasjoner og dårligst Rating

App	Category	Installasjoner	Rating
Google Play Movies & TV	VIDEO_PLAYERS	1000000000	3.7
Google Play Books	BOOKS_AND_REFERENCE	100000000	3.9
Google News	NEWS_AND_MAGAZINES	1000000000	3.9
Messenger – Text and Video Chat for Free	COMMUNICATION	1000000000	4.0
Hangouts	COMMUNICATION	1000000000	4.0

```
print('Apper sortert etter Størrelse og flest Installasjoner')
display(df.sort_values(by=['Størrelse', 'Installasjoner'], \
ascending=[False, False]) \
[[['App', 'Category', 'Størrelse', 'Installasjoner']] \
.head(3).set_index('App'))
```

✓ 0.0s

Apper sortert etter Størrelse og flest Installasjoner

App	Category	Størrelse	Installasjoner
Hungry Shark Evolution	GAME	100000000.0	100000000
SimCity BuildIt	FAMILY	100000000.0	50000000
Miami crime simulator	GAME	100000000.0	10000000

```
siste_oppdatering = df.sort_values(by='Sist oppdatert', \
ascending=False)[['Sist oppdatert']].iloc[0][0].date()
print(f'Dato for siste oppdatering: {siste_oppdatering}')
```

✓ 0.1s

Dato for siste oppdatering: 2018-08-08

### Grupperinger

#### Grupperinger

##### Kategorier med flest apper

```
display((df.groupby('Category').size() \
[[['Category']]).sort_values(ascending=False)).head(3)
```

✓ 0.3s

Category	
FAMILY	1873
GAME	945
TOOLS	829
dtype: int64	

##### Apper med flest anmeldelser i hver kategori

```
display(df.loc[df.groupby('Category')[['Reviews']].idxmax() \
[[['Category', 'App', 'Reviews']] \
.sort_values(by='Reviews', ascending=False) \
[[['Category', 'App', 'Reviews']] \
.head(5).set_index('Category'))
```

✓ 0.1s

Category	App	Reviews
SOCIAL	Facebook	78158306
COMMUNICATION	WhatsApp Messenger	69119316
GAME	Clash of Clans	44893888
TOOLS	Clean Master- Space Cleaner & Antivirus	42916526
VIDEO_PLAYERS	YouTube	25655305

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>



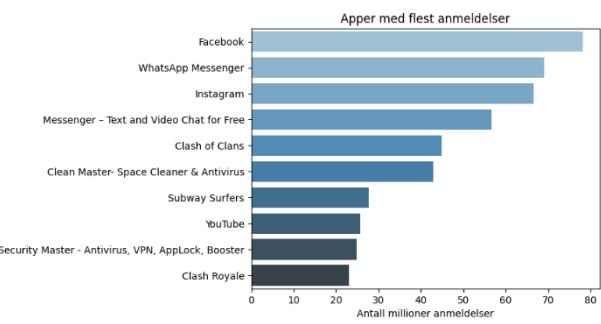
## Diagrammer

### Diagrammer

#### Topp 10 apper med flest anmeldelser

```
df_sort_review = df.sort_values(by='Reviews', ascending=False)
df_sort_review['Reviews'] = df_sort_review['Reviews']/1000000
sns.reset_defaults()
ax = sns.barplot(x='Reviews', y='App',
                  data=df_sort_review.head(10),
                  orient='h', palette='Blues_d')
ax.set(title='Apper med flest anmeldelser')
ax.set(xlabel='Antall millioner anmeldelser', ylabel='')
plt.show()
```

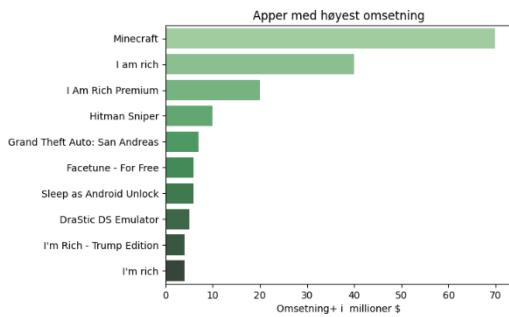
0.6s



#### Topp 10 apper med høyest omsetning

Tallene er kun grove estimater på grunn av begrenset nøyaktighet i antall installasjoner (kun størrelsesorden)

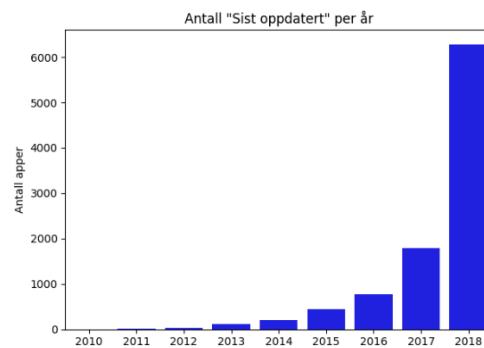
```
df_omsetning = df[df['Type'] == 'Paid'].copy()
df_omsetning['Omsetning'] = (df_omsetning['Price'] *
                             df_omsetning['Installasjoner'])/1000000
df_omsetning = df_omsetning.sort_values(by='Omsetning',
                                         ascending=False).head(10)
sns.reset_defaults()
ax=sns.barplot(x='Omsetning', y='App',
                 data=df_omsetning,
                 orient='h', palette='Greens_d')
ax.set(title='Apper med høyest omsetning')
ax.set(xlabel='Omsetning+ i millioner $', ylabel='')
plt.show()
```



### Eksempler med Sist oppdatert

Det ser ut som de fleste appene oppdateres årlig.

```
import calendar
import locale
locale.setlocale(locale.LC_ALL, 'nb_NO.UTF-8')
df['År sist oppdatert'] = df['Sist oppdatert'].dt.year
sns.reset_defaults()
ax = sns.barplot(data = df['År sist oppdatert'] \
|.....value_counts().sort_index().reset_index(),
                  x='År sist oppdatert', y='count', color='blue')
ax.set(title='Antall "Sist oppdatert" per år')
ax.set(xlabel='År', ylabel='Antall apper')
plt.tight_layout()
plt.show()
```



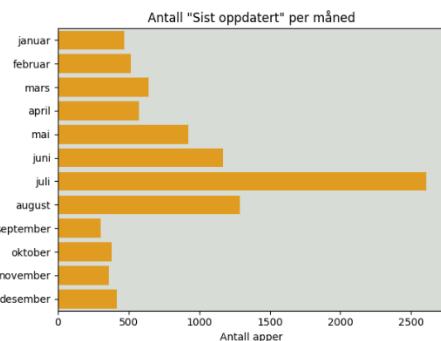
Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Det oppdateres flest apper i juli

```
df['Måned sist oppdatert'] = df['Sist oppdatert'].dt.month
s_month = df.groupby('Måned sist oppdatert').size()
s_month.index = list(calendar.month_name)[1:]
sns.reset_defaults()
ax = sns.barplot(x=s_month, y=s_month.index,
                  color='orange', orient='h')
ax.set(title='Antall "Sist oppdatert" per måned')
ax.set_xlabel('Antall apper', ylabel='')
ax.set_facecolor('xkcd:light grey')
plt.tight_layout()
plt.show()
```

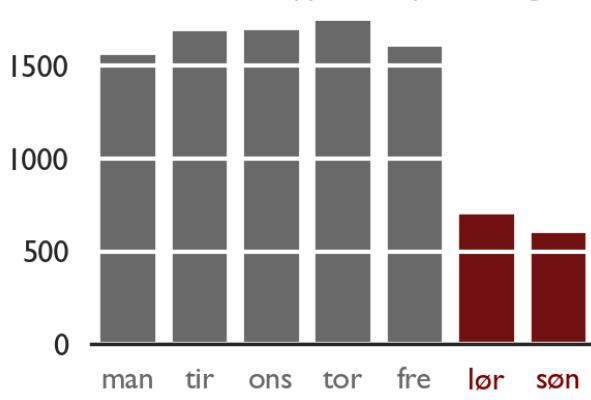


Det oppdateres færre apper i helgene.

Design inspirert av E.Tufte

```
df['Ukedag sist oppdatert'] = df['Sist oppdatert'].dt.weekday
s_weekday = df.groupby('Ukedag sist oppdatert').size()
s_weekday.index = [day[:3] for day in calendar.day_name]
rc = {
    "font.family": "Gill Sans MT",
    "axes.titlesize": 20,
    "xtick.labelcolor": "dimgrey",
}
palette = 5*["dimgrey"] + 2*["#800000"]
sns.reset_defaults()
sns.set_theme(style="white", context="poster",
              rc=rc, palette=palette)
ax = plt.gca()
ax = sns.barplot(x=s_weekday.index, y=s_weekday,
                  orient='v', zorder=0, palette=palette)
ax.yaxis.set_major_locator(MultipleLocator(500))
ax.yaxis.grid(color="white", linewidth=3)
ax.get_xticklabels()[5].set_color("#800000")
ax.get_xticklabels()[6].set_color("#800000")
ax.set(title='Antall "Sist oppdatert" per ukedag')
sns.despine(bottom=False, left=True)
plt.tight_layout()
plt.show()
```

Antall "Sist oppdatert" per ukedag



Datarensingen og datautforskingen som er gjort i de forgående avsnittene, kunne vært utført på forberedelsesdagen. I så fall, ville elevene kunne spart mye tid på eksamen.

### Ta med minst dette

I denne bolken har vi lært å rense og utforske data. Det vil si å bli kjent med dataene ([head](#), [info](#)), rette feil ([fillna](#), [replace](#)), fjerne duplikater ([drop\\_duplicates](#)), håndtere ulike formater ([apply](#), [pd.to\\_datetime](#)), og manglende verdier ([isna](#), [dropna](#), [fillna](#)). Vi har også sett på hvordan vi kan utforske dataene for å forstå sammenhenger og mønstre ([groupby](#), [sort\\_values](#)). Videre har vi brukt enkle sentral- og spredningsmål for å analysere dataene ([describe](#)). Dette er viktige ferdigheter for å kunne innhente, analysere og presentere informasjon fra reelle datasett, slik det kreves i IT-2.

### Øvingsoppgaver

#### 4.3.1

Gjør deg kjent med datasettet som ble gitt på forberedelsesdagen til IT-2 eksamen våren 2024. Bruk det du har lært om datarensing og datautforskning i denne bolken

#### "Vedlegg 2 – datasett med tidsbruk på ulike aktiviteter

Vedlagt ligger det et datasett som inneholder en oversikt over tidsbruk på ulike aktiviteter, i CSV- og JSON-format. Du kan velge hvilket av formatene du vil bruke. Datasettet kan lastes ned ved [å følge denne lenken](#).

Vurderingsseksemplar

## **Smidig IT-2**

### **for eksklusiv bruk ved <navn på skole>**

Datasettet inneholder tid brukt til ulike aktiviteter en gjennomsnittsdag året 2000 kategorisert etter aktiviteter og kjønn.

De ulike aktivitetene er gruppert i kategorier hvor tidsbruken er summert for kategoriene. Aktivitetene under hver kategori er kodet med innrykk.

Tidsbruk er angitt i hele timer og minutter skilt med punktum. Merk at dette ikke er timer med desimaltall.

Datasettet er kodet med tegnsettet UTF-8. Skilletegnet i CSV-filen er semikolon.

Last ned datasettet i ønsket format og gjør deg kjent med det. Du må også forberede deg på å behandle datasettet med programmering og tilpasse data i settet for å kunne gjøre de aktuelle beregningene. I tillegg må du kontrollere at du har de nødvendige ressursene installert på den datamaskinen du skal bruke på eksamen. På eksamenen vil du få to oppgaver hvor du skal behandle dette datasettet og presentere informasjon fra det med programmering."

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 4.4 Datasimuleringer

#### Bolkens innhold

Innledning .....	267
Skrått kast (Fysikk).....	267
Økosystem (Biologi) .....	268
Lyskryss (Informatikk).....	269
Ta med minst dette.....	272
Øvingsoppgaver.....	272

#### Kompetansemål:

- anvende objektorientert modellering til å beskrive et programs struktur
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- utforske og vurdere alternative løsninger for design og implementering av et program

#### Innledning

Datasimulering er en metode hvor vi bruker datamaskiner til å lage modeller av virkelige systemer for å forstå hvordan de oppfører seg under forskjellige forhold. I stedet for å utføre eksperimenter i virkeligheten, som kan være dyrt eller tidkrevende, kan vi simulere dem for å få kostnadseffektive resultater raskt. Tenk bare på flysimulatorer. Datasimulering brukes i flere bransjer som meteorologi (for å forutsi været), finans (for å modellere markedsrisiko) og helse (for å simulere spredning av sykdommer). Innen statistikk kan direkte beregninger være så komplekse at vi bruker simuleringer som gir tilfredsstillende resultater i stedet.

Vi har allerede jobbet med simuleringer da vi lærte om [sett](#) i bok [3.3 Sett og boolsk algebra](#), som i [oppgave 3.3.1](#) med Yatzy-kast. I denne bolken skal vi utforske flere eksempler for å få en bedre forståelse av hvordan vi kan lage datasimuleringer. La oss starte med et enkelt, men illustrerende eksempel.

#### Skrått kast (Fysikk)

Tenk deg at det ligger en ball på bakken. Hvor langt vil den fly uten luftmotstand før den treffer bakken igjen dersom vi sparker til den så den får en utgangshastighet på 25 m/s i en 30° graders vinkel? Hvis vi ikke klarer å regne det ut, kan vi bruke datasimulering for å finne svaret. Siden gravitasjonen er den eneste kraften som virker på ballen, vil ballens posisjon være gitt ved

$$s_x = v_{0x} \cdot t \quad s_y = v_{0y} \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

Hvor



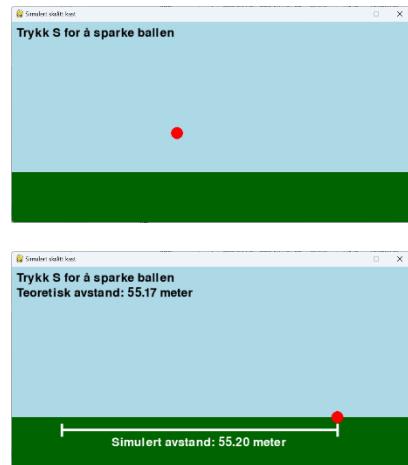
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

$s$  er avstand,  $v_0$  er utgangshastighet,  $t$  er tiden som øker med  $\frac{1}{FPS}$  for hvert bilde,  $g$  er gravitasjon ( $9.81 \frac{m}{s^2}$ ) og  $x$  og  $y$  er komponentene i x- og y-retning. Hvis vi følger med på når y-posisjonen igjen er på bakkenivå, kan vi lese av x-posisjonen for å se hvor langt ballen har flydd.

Hvis vi likevel vil regne på det, må vi finne ut hvor lenge ballen er i lufta. Dette er dobbelt så lenge som tiden det tar for ballen å nå toppen, som vi kan finne ved å derivere  $s_y(t)$  med hensyn på  $t$  og  $s_y'(t) = 0$ . Se filen

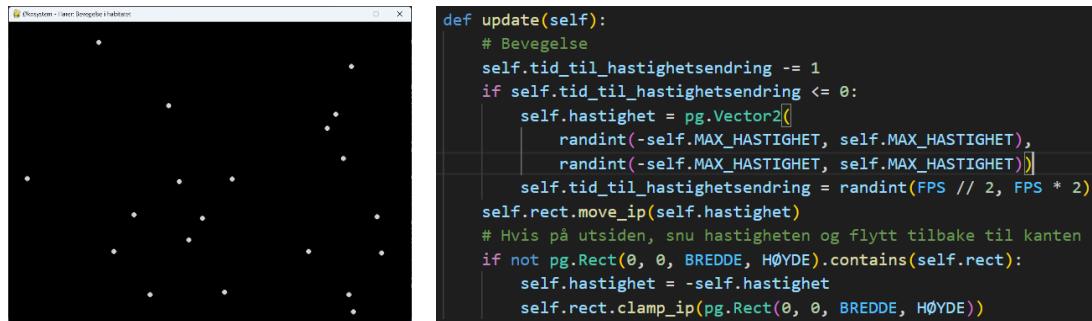
[b\\_4\\_4\\_1\\_skrått\\_kast.py](#).



### Økosystem (Biologi)

Økologiske simuleringer hjelper oss å forstå hvordan populasjoner av ulike arter påvirker hverandre i naturen. I dette eksempelet skal vi simulere et økosystem som består av harer og rever. Vi vil bruke ulike biologiske og økologiske prinsipper for å modellere bevegelse, livslengde, reproduksjon, bæreevne og utvikling over tid og tar utgangspunkt i den objektorienterte malen for Pygame, [b\\_3\\_4\\_01\\_pygame\\_mal.py](#).

**Harer - Bevegelse i habitatet:** Vi starter med 20 harer som beveger seg med tilfeldig hastighet som endres ved tilfeldige intervaller. Dersom en hare forsøker å forlate habitatet, endrer den retning for å forblie innenfor området. Se [b\\_4\\_4\\_2\\_1\\_habitat.py](#).

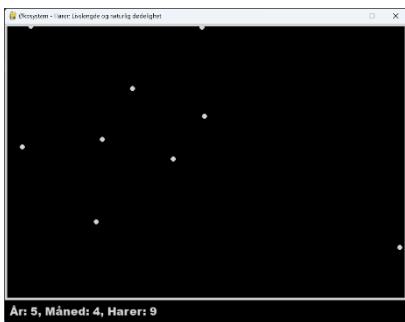


**Harer - Livslengde og naturlig dødelighet:** En hare kan leve opp til 14 år under ideelle forhold, men i virkeligheten blir de ofte ikke mer enn 5 år gamle. For å simulere naturlig dødelighet, antar vi at en hare har følgende sannsynlighet for å dø hver måned, som øker med alderen:

$$\text{sannsynlighet} = (\text{alder i måneder} / \text{maksimal levetid i måneder})^3$$

Vi lar 1 sekund simuleringsstid tilsvare 1 måned og viser en statuslinje med år, måned og antall harer i habitatet til enhver tid. Se filen [b\\_4\\_4\\_2\\_2\\_habitat.py](#).

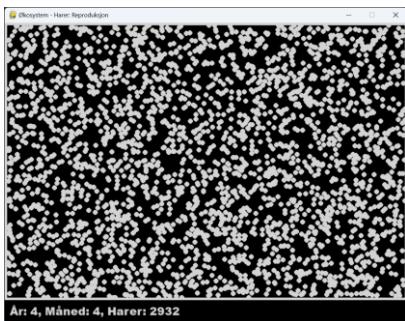
## Smidig IT-2 for eksklusiv bruk ved <navn på skole>



```
# Alder
self.alder = self.habitat.måneder - self.fødselsmåned
# Død
if self.dør():
    self.kill()

def dør(self):
    # sjekker om det er en ny måned og beregner sannsynligheten for død
    return (
        self.habitat.tid % FPS == 0
        and random() < (self.alder / self.MAKS_ALDER) ** 3
    )
```

**Harer - Reproduksjon:** Harer blir forplantningsdyktig etter 1 år og får normalt 2 kull per år, med 2 til 7 unger per kull. Vi antar at det er 40% sannsynlighet for at hver kjønnsmoden hare får et kull hver sjette måned. Se filen [b\\_4\\_4\\_2\\_3\\_habitat.py](#).

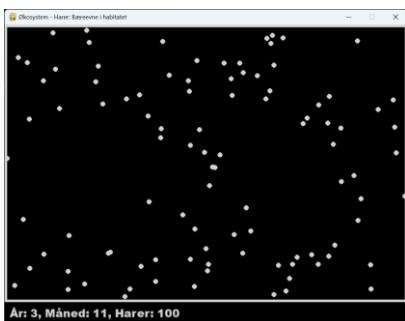


```
# Reproduksjon
if self.får_kull():
    for _ in range(randint(self.MIN_UNGER_i_KULLET, self.MAX_UNGER_i_KULLET)):
        self.habitat.harer.add(
            Hare(self.habitat, self.rect.x, self.rect.y))

def dør(self): ...

def får_kull(self):
    # sjekker om det er en ny måned og om det er tid for å få kull
    return (
        self.habitat.tid % FPS == 0
        and self.alder >= self.KJØNNSMODEN_ALDER
        and self.alder % self.MÅNEDER_MELLOM_KULL == 0
        and random() < self.SJANSE_FOR_Å_FÅ_UNGER
    )
```

**Harer - Bæreevne i habitatet:** Habitatet har ikke bæreevne til mer enn 100 harer. Dersom antallet harer overskriver bæreevnen, har hver hare 20% sannsynlighet for å dø hver måned. Dersom antallet harer i habitatet er under bæreevnen, dør de kun av alderdom. Se filen [b\\_4\\_4\\_2\\_4\\_habitat.py](#).

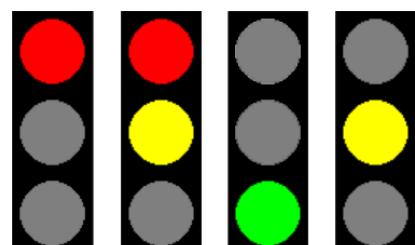


```
def dør(self):
    # sjekker om det er en ny måned og beregner sannsynligheten for død
    bæreevne = self.habitat.BÆREEVNE_HARER
    return (
        self.habitat.tid % FPS == 0 and (
            self.habitat.antall_harer > bæreevne and random() < 0.2) or
            (random() < (self.alder / self.MAKS_ALDER) ** 3)
    )
```

Vi har brukt en smidig tilnærming, startet enkelt og gradvis utvidet modellen for å gjøre den mer realistisk. Denne fremgangsmåten viser hvordan smidig utvikling også kan anvendes i mindre programmeringsoppgaver, ikke bare i store systemutviklingsprosjekter.

### Lyskryss (Informatikk)

Vi skal simulere trafikkstyring i et veikryss med trafikklys. Før vi introduserer bilene, koder vi trafikklyset som har fire tilstander som går i syklus: rød, rød\_og\_gul, grønn og gul. Varighetene er satt til henholdsvis 7 sekunder for rød og



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

grønn, og 3 sekunder for rød\_og\_gul og bare gul. Den nåværende tilstanden bestemmer hvilke av lysene som skal lyse.

```
class Trafikklys(pg.sprite.Sprite):
    def __init__(self, posisjon):
        super().__init__()
        self.image = pg.Surface((50, 150))
        self.image.fill(pg.Color(FARGER['trafikklys']))
        self.rect = self.image.get_rect(center=posisjon)
        self.lys_syklus = deque([{'rød': 7 * FPS}, {'rød_og_gul': 3 * FPS},
                               {'gul': 3 * FPS}, {'grønn': 7 * FPS}])
        self.tilstand = ''
        self.tid_igjen = 0
        self.sett_tilstand()
```

Tilstanden er nøkkelen og varigheten er verdien i ordbøkene som ligger i en `deque` med navnet `self.lys_syklus`. Vi bruker `deque` i stedet for `list`, for da kan vi rotere elementene. Dermed unngår vi bruke en indeks til å holde rede på den nåværende tilstanden,

for vi kan alltid ha den i posisjon 0. `self.tilstand` er gjeldende tilstand, og `self.tid_igjen` er en teller som angir hvor lenge til den gjeldende tilstanden varer. Til slutt i `__init__()` og hver gang tilstanden endres, kalles `sett_tilstand()`.

Ordboken med informasjon om gjeldende tilstand ligger først i `self.lys_syklus`. Når programmet starter og etter hvert

rotering (oppdatering av tilstanden) hentes den første ordboken, og `self.tilstand` og `self.tid_igjen` settes til henholdsvis nøkkelen og verdien. Deretter

kalles `oppdater_lys()` for å slå på de riktige lysene for den nye tilstanden.

```
def sett_tilstand(self):
    ordbok = self.lys_syklus[0]
    self.tilstand = list(ordbok.keys())[0]
    self.tid_igjen = ordbok[self.tilstand]
    self.oppdater_lys()
```

```
def oppdater_lys(self):
    lys = [('rød', (25, 25), 'red'),
           ('gul', (25, 75), 'yellow'),
           ('grønn', (25, 125), 'green')]
    for farge, posisjon, farge_kode in lys:
        lys_farge = farge_kode if farge in self.tilstand else 'gray50'
        pg.draw.circle(self.image, pg.Color(lys_farge), posisjon, 20)
```

Informasjon om lysene ligger i listen `lys`, hvor hvert element er en tuppel med informasjon om `farge`, `posisjon` og `farge_kode`. `farge` har de samme ordene som brukes i tilstanden, så for

tilstanden 'rød\_og\_gul' vil både det røde og det gule lyset slås på. `farge_kode` er navnet som brukes av Pygame. Først i løkken bestemmes det om `lys_farge` skal være `farge_kode` eller grå ('gray50'), og deretter tegnes lyset med riktig farge i riktig posisjon.

`self.tid_igjen` oppdateres for hvert bilde. Vi har valgt å kalle metoden `update` slik at den kalles automatisk så lenge trafikklyset er en `Sprite`, og vi har lagt den inn i spritegruppen `all_sprites` i `App`. Når `all_sprites.update()` kalles fra `oppdater()` i hovedløkken, kjøres trafikklysets `update()` automatisk. Når tiden er ute, roterer vi elementene i `self.lys_syklus`. Alle tilstandene flyttes én posisjon fram, og den første tilstanden flyttes bakerst. Når vi nå kaller `self.sett_tilstand()`, vil denne metoden bruke informasjon om neste tilstand som nå står i posisjon 0.

```
def update(self):
    self.tid_igjen -= 1
    if self.tid_igjen <= 0:
        self.lys_syklus.rotate(-1)
        self.sett_tilstand()
```

Se filen `b_4_4_3_1_lyskryss.py`.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
def __init__(self):
    self.ankomst_rate = 1/3 # Hvert 3. sekund
    self.planlegg_neste_ankomst()

def planlegg_neste_ankomst(self):
    ankomst_tid_ms = int(random.expovariate(self.ankomst_rate) * 1000)
    pg.time.set_timer(BIL_ANKOMST_EVENT, ankomst_tid_ms)

def håndter_hendelser(self):
    for hendelse in pg.event.get():
        if hendelse.type == pg.QUIT:
            self.kjører = False
        elif hendelse.type == BIL_ANKOMST_EVENT:
            self.legg_til_ny_bil()
            self.planlegg_neste_ankomst()
```

Vi er nå klare til å introdusere biler og bruker `random.expovariate()`<sup>27</sup> for å simulere realistiske intervaller mellom ankomster. Vi oppgir kun én verdi, som er 1 delt på ønsket gjennomsnittstid mellom ankomster, og setter det hele i gang ved å kalle `planlegg_neste_ankomst()` fra `__init__()`. Her bruker vi `pg.time.set_timer()`, som planlegger når en bestemt hendelse skal skje i fremtiden. `BIL_ANKOMST_EVENT` er en brukerdefinert hendelse, `pg.USEREVENT + 1`. Denne hendelsen fanges opp i `håndter_hendelser()`, som legger til en ny bil og kaller `planlegg_neste_ankomst()` på nytt, og slik fortsetter det. Se også Alternativ 3 i [Tidsstyrte handlinger](#) i 4.2 Hendelser, kollisjoner, tidsstyring og GUI-integrering i Pygame.

```
def legg_til_ny_bil(self):
    ny_bil = Bil(self)
    self.juster_bil_posisjon(ny_bil)
    self.biler.add(ny_bil)
    self.alle_sprites.add(ny_bil)

def juster_bil_posisjon(self, ny_bil):
    # i tilfelle det er en bil der fra før så den ikke overlapper
    if self.biler:
        siste_bil = max(self.biler, key=lambda bil: bil.rect.top)
        ny_bil.rect.top = max(
            ny_bil.rect.top, siste_bil.rect.bottom + MINSTE_AVSTAND)
```

Vi er nå klare til å introdusere biler og bruker `random.expovariate()`<sup>27</sup> for å simulere realistiske intervaller mellom ankomster. Vi oppgir kun én verdi, som er 1 delt på ønsket gjennomsnittstid mellom ankomster, og setter det hele i gang ved å kalle `planlegg_neste_ankomst()` fra `__init__()`. Her bruker vi `pg.time.set_timer()`, som planlegger når en bestemt hendelse skal skje i fremtiden. `BIL_ANKOMST_EVENT` er en brukerdefinert hendelse, `pg.USEREVENT + 1`. Denne hendelsen fanges opp i `håndter_hendelser()`, som legger til en ny bil og kaller `planlegg_neste_ankomst()` på nytt, og slik fortsetter det. Se også Alternativ 3 i [Tidsstyrte handlinger](#) i 4.2 Hendelser, kollisjoner, tidsstyring og GUI-integrering i Pygame.

Bilene legges til rett under skjermen og beveger seg oppover, mot nord. Dersom bilene ankommer så hyppig at de ville bli plassert oppå hverandre, justerer vi posisjonen ved å plassere den nye bilen `MINSTE_AVSTAND` bak den bakerste bilen.

```
def oppdater(self, trafikklys, biler):
    nåværende_lys = trafikklys.nåværende_tilstand
    stoppet = False
    i_kryss = self.rect.colliderect(KRYSS_REKTANGEL)

    # Stopp hvis bilen kommer fram til krysset og lyset ikke er grønt
    kryss_sjekk_rekt = self.rect.move(self.fart)
    if not i_kryss:
        stoppet = (nåværende_lys != 'grønn') and \
                  kryss_sjekk_rekt.colliderect(KRYSS_REKTANGEL)
```

Bilene skal reagere på trafikklys kun når de er i ferd med å kjøre inn i et kryss. Det skjer når bilens neste posisjon vil føre til en kollisjon med krysset (`KRYSS_REKTANGEL`), og da skal bilen stoppe hvis lyset ikke er grønt. Alle andre steder skal bilen fortsette å kjøre framover med ett unntak:

```
# Hold avstand til bilen foran (som venter på grønt lys)
avstand_sjekk_rekt = self.rect.move(0, -MINSTE_AVSTAND)
if not stoppet:
    stoppet = any(bil != self and avstand_sjekk_rekt.colliderect(
        bil.rect) for bil in biler)

# Hvis ikke stoppet, flytt bilen fremover
if not stoppet:
    self.rect.move_ip(self.fart)

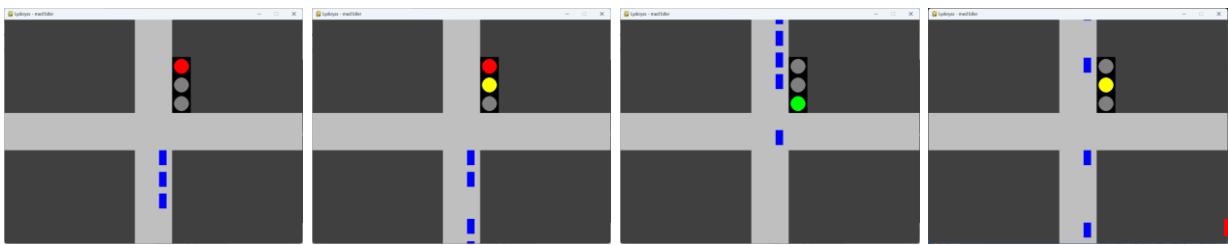
# Fjern bilen hvis den har kjørt ut av skjermen
if self.rect.bottom < 0:
    self.kill()
```

Dersom det allerede står biler i kø foran krysset, må bilen stoppe bak den bakerste bilen i køen. Det får vi til ved å sjekke om bilen vil kolidere med en annen bil dersom den hadde beveget seg `MINSTE_AVSTAND` framover. Se filen [b\\_4\\_4\\_3\\_2\\_lyskryss.py](#).

<sup>27</sup> Poisson-fordeling med negativ eksponentialfordeling mellom ankomster.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>



### Ta med minst dette

Datasimulering er en kostnadseffektiv metode for å modellere virkelige systemer ved hjelp av datamaskiner. Vi har anvendt noe av de vi har lært fra tidligere bolker til å utforske tre eksempler: Skrått kast (fysikk), Økosystem (biologi) og Lyskryss (informatikk). simuleringen av lyskrysset viste vi hvordan vi kan rotere elementene (tilstandene) i en `deque`, i stedet for å holde styr på en indeks i en liste.

### Øvingsoppgaver

#### 4.4.1

Lag en datasimulering i Pygame som finner bremselengden til en bil som kjører i 80 km/t ( $v_0$ ) på våt asfalt hvor friksjonskoeffisienten er 0.7 ( $\mu$ ). Vi antar en reaksjonstid på 1 sekund før føreren begynner å bremse. Bruk formelene for akselerasjon og bevegelse for å beregne og visualisere bilens bremselengde. Les av  $s$  når  $v = 0$ .

$$s = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \quad v = v_0 + a \cdot t \quad a = -\mu \cdot g$$

Hvor

$s$  er avstand

$v_0$  er utgangshastighet (80 km/t)

$t$  er tid

$a$  er akselerasjon

$v$  er hastighet

$\mu$  er friksjonskoeffisient (0.7)

$g$  er gravitasjon (9.81 m/s<sup>2</sup>)

#### 4.4.2

Ta utgangspunkt i filen [b\\_4\\_4\\_2\\_4\\_habitat.py](#) og utvid modellen som forklart nedenfor.

a)

**Rever - Populasjon, livslengde, reproduksjon og bæreevne:** Legg til en revebestand i habitatet, tilsvarende harebestanden, men med følgende verdier:

- Start med 4 rever som beveger seg tilsvarende harene, men 25% langsommere. Gjør revene litt større enn harene og gi dem en annen farge.
- En rev kan leve opp til 12 år under ideelle forhold, men i virkeligheten blir de ofte ikke mer enn 3 år gamle. Anta at en rev har følgende sannsynlighet for å dø hver måned, som øker med alderen:

$$\text{sannsynlighet} = (\text{alder i måneder} / \text{maksimal levetid i måneder})^2$$

Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

- Rever blir forplantningsdyktig etter 10 måneder og får normalt 1 kull per år, med 3 til 9 unger per kull. Anta at det er 40% sannsynlighet for at hver kjønnsmoden rev får et kull hvert år.
- Habitatet uten harer har en grunnbæreevnne til 5 rever, men bæreevnen øker med antall harer etter formelen

$$\text{justert bæreevne} = \text{grunnbæreevne} * \left( 1 + 0,4 * \left( \frac{\text{antall harer i habitatet}}{\text{bæreevnen for harer}} \right) \right)$$

Dersom antallet rever overskridet bæreevnen, har hver rev 20% sannsynlighet for å dø hver måned. Dersom antallet rever i habitatet er under bæreevnen, dør de kun av alderdom.

**b)**

**Habitat - Predasjon:** Anta at når en rev treffer en hare, blir haren spist (dør).

4.4.3

**a)**

Ta utgangspunkt i filen [b\\_4\\_4\\_3\\_1\\_lyskryss.py](#) og utvid modellen slik at det blir fire trafikklys, ett for hver kjøreretning. Lysene i nord-sør retning skal lyse synkront, og lysene i øst-vest retning skal være to tilstander forskjøvet. Dette betyr at øst-vest har rødt lys når nord-sør har grønt lys. For å oppnå dette kan det være lurt å flytte styringen ut av [Trafikklys](#)-klassen og inn i en ny Trafikklyskontroller-klasse. Trafikklysene vil da bare vise riktig lys basert på signaler fra kontrolleren.

**b)**

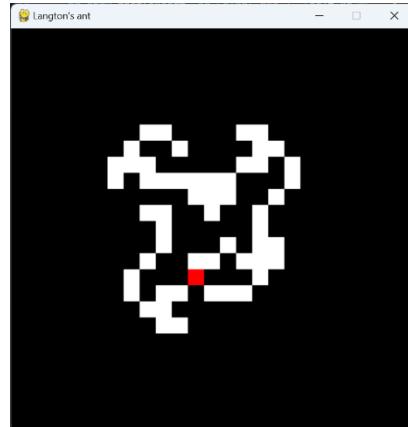
Se på [b\\_4\\_4\\_3\\_2\\_lyskryss.py](#) og utvid modellen med biler som kjører i alle retninger.

4.4.4

**a)**

Tenk deg en kunstner som beveger seg rundt i et rom med svarte fliser lagt ut i et rutenett. Kunstneren har på seg en rød jakke og starter midt i rommet med en tilfeldig retning (nord, sør, øst eller vest). Han ser på flisen han står på og følger disse reglene:

1. Er flisen svart:
  - Maler kunstneren den hvit.
  - Dreier 90° mot venstre.
  - Går én flis frem i den nye retningen.
2. Er flisen hvit:
  - Maler kunstneren den svart.
  - Dreier 90° mot høyre.
  - Går én flis frem i den nye retningen..
3. Vegger:
  - Hvis kunstneren prøver å gå gjennom en vegg, kommer han inn fra motsatt side av rommet.

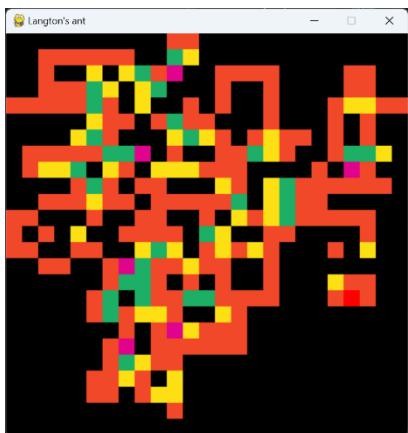


## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

Simuler denne kunstnerens vandring og la det gå 50 ms mellom hvert steg.<sup>28</sup>

**b)**

Utvid modellen med flere farger, flere regler og tilfeldighet.



### 4.4.5

Oppgave 10: Livets spill<sup>29</sup>, [IT-2 Eksamens Vår 2024](#).

I denne oppgaven skal du lage et program som simulerer livssyklusen til celler, altså hvordan celler oppstår, lever og dør, og hvordan én generasjon med celler gir opphav til en ny.

Programmet skal bygges opp som et todimensjonalt rutenett med celler som alle kjenner til sine naboceller. Cellene kan veksle mellom tilstandene levende og død. Celler reproduseres eller dør basert på de fire følgende regler for liv og død:

- Enhver levende celle med færre enn to levende naboyer dør, som ved underpopulasjon.
- Enhver levende celle med to eller tre naboyer lever videre til neste generasjon.
- Enhver levende celle med flere enn tre levende naboyer dør, som ved overpopulasjon.
- Enhver død celle med akkurat tre levende naboyer blir vekket til live, som ved reproduksjon.

På de andre arkfanene ser du eksempler på hvordan en generasjon celler blir til neste generasjon celler når reglene ovenfor gjelder.

Programmet skal utvikles som et objektorientert program med nødvendige klasser og metoder. Du kan svare på oppgaven stegvis eller som ett samlet program, akkurat som du selv ønsker. Ta utgangspunkt i disse stegene:

- a) Lag et spillebrett med et startoppsett av levende og døde celler, såkalt generasjon 0, der hver celle på spillebrettet har 1/3 sjanse til å starte som levende.
- b) Implementer muligheter for å
  - endre alle cellene til døde, slik at du får et «tomt» brett
  - bytte om statusen til en enkeltcelle fra død til levende og omvendt, ved å klikke på den

<sup>28</sup> [Langton's ant](#)

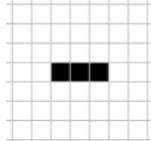
<sup>29</sup> [Game of Life](#)

Vurderingsseksemplar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

- c) Utvid programmet med funksjonalitet som automatisk oppdaterer tilstanden til hver celle i spillebrettet fra gjeldende generasjon til den neste, basert på de fire reglene som er beskrevet ovenfor. Alle cellene skal oppdateres samlet.

Om man starter med følgende levende celler som generasjon 0,



vil den etterfølgende serien med generasjoner bli slik:

Generasjon 1	Generasjon 2	Generasjon 3	Generasjon 4	Generasjon 5

## 4.5 Standarder for datahåndtering

### Bolkens innhold

Innledning .....	276
Standarder for lagring av ulike typer data.....	276
Standarder for utveksling av ulike typer data.....	277
Standarder for sikring av ulike typer data.....	279
Ta med minst dette.....	282
Øvingsoppgaver.....	282

### Kompetansemål:

- gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data

### Innledning

Å gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data er en omfattende oppgave. Det finnes hundrevis av slike standarder, og hver av dem er vanligvis på hundrevis av sider. Vi vil derfor begrense oss til kortfattede beskrivelser av de mest sentrale standardene for datahåndtering.

I tidligere bolker har vi sett på noen standarder for lagring og utveksling av ulike typer data, blant annet i [1.6 Filer](#), [1.9 Reelle datasett med CSV](#) og [2.7 Utveksling av data med JSON](#), og vi vil presentere REST-standarden for utvikling av nettverkstjenester [i 4.9 Reelle datasett med REST API](#). Før vi går videre til standarder for sikring av ulike typer data, vil vi oppsummere og utvide det vi har lært om standarder for lagring og utveksling.

### Standarder for lagring av ulike typer data

I bok 1.6 Filer så vi at data i form av tekst lagres som tegn på 8 bit (1 byte) bestemt av karakterkodingsstandarden for tegnsettet som brukes. For å benytte norske karakterer har vi tatt i bruk [utf-8](#), [iso-8859-1 \(latin-1\)](#) og [windows-1252](#) som argumenter til [encoding](#)-parameteren når har lest fra og skrevet til filer. Tall blir lagret på en annen måte, og programmet må vite om dataene skal tolkes som eksempelvis tekst, heltall eller desimaltall. Derfor snakker vi også om ulike datatyper. Det er forskjell på tegnet "1" og talet 1.

Det finnes standarder for hvordan innholdet i ulike filtyper skal lagres, og det er også hundrevis av slike standarder. I tillegg finnes det en forskrift som bestemmer hvilke [formater som skal brukes i offentlige arkiver](#). Den kan



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

vi lese om hvilke standarder som er tillatt for ulike typer data som dokumenter, fotografier, kart, video, lyd, med mer. Eksempelvis er TIFF og PDF/A tillatt for tekst med objekter.

Vanligvis blir data lagret i databaser, og da blir relasjonsdatabase (RDBMS)- og NoSQL-standardene ofte brukt. Når data fra disse systemene skal overføres til andre systemer, er det vanlig å bruke standarder som JSON og XML.

For å kunne lagre data i datamaskiner, er det nødvendig å ha et lagringsmedium. I dag er det vanlig å bruke harddisker (HDD) eller SSD-er (Solid State Drives). Disse lagringsenhetene har sine egne standarder for grensesnitt og overføringshastighet, for eksempel SAS, SATA og PCIe. Før disse lagringsenhetene kan brukes, må de formateres for å kunne organisere dataene som skal lagres. Dette gjøres ved hjelp av standarder for filsystemer, som for eksempel FAT, NTFS (Windows), EXT4 (Unix) og APFS (MacOS). Forskjellige operativsystemer har ulike krav til filsystemer og formatering, så det er viktig å velge riktig filsystem og formatere riktig når vi arbeider med data på tvers av forskjellige plattformer.

I fremtiden vil vi sikkert se nye typer lagringsmedier og filformater, samtidig som gamle teknologier går ut på dato, som for eksempel magnetbånd, mikrofilm, floppy-disker, CD-er og DVD-er, eller BMP (bildefiler), WPD (WordPerfect dokumenter) og WK1 (Lotus-1-2-3 regneark). Å overføre data fra ett lagringsmedium til et annet eller konvertere data fra ett format til et annet, er en stor utfordring. I 2017 var det bevart 551 kilometer arkivmateriale i Norge. "Satt i tettpakkede hyller stilt etter hverandre fyller det bilveien fra Sande i Vestfold gjennom Østerdalen og helt til Trondheim"<sup>30</sup>. Mengden informasjon som lagres øker raskt, og bare i 2021 digitaliserte Norsk Helsearkiv alene 36 millioner journalsider, noe som tilsvarer 11 ganger høyden på Eiffeltårnet<sup>31</sup>. Det er derfor viktig å sikre at vi ikke mister tilgang til viktig informasjon når teknologien endrer seg. For å opprettholde tilgang til eldre data, er det nødvendig å kontinuerlig overføre og konvertere data til nye formater og lagringsmedier, samt lagre informasjon om hvordan dataene skal tolkes og åpnes.

#### **Standarder for utveksling av ulike typer data**

Når data skal overføres fra en datamaskin til en annen er det mange elementer, både maskinvare og programvare, som må snakke med hverandre på samme språk. Vi har sett at data kan overføres på for eksempel JSON- og XML format, og vi har lastet ned CSV-filer. Formatet eller typen data er mindre viktig for selve utveksling eller overføringen. Det spiller ingen rolle om innholdet i filen er en video, et bilde, et dokument eller lyd.

Kommunikasjonsstandardene er sinnrike protokoller satt i et lagdelt system. En protokoll er regler for hvordan informasjonen skal utveksles mellom et lag hos senderen og tilsvarende lag hos mottakeren. Det er mye som må være avtalt og tilpasset for at en overføring av data skal være vellykket, som den fysiske formen på pluggen så den passer i kontakten, antall ledere i kabelen, rollen til hver enkelt leder, signalstyrke, frekvens, hvordan *bit* skal sendes og settes sammen til *bytes*. Videre inkluderer standardene også metoder for feilhåndtering, adressering, ruting og annen nettverksfunksjonalitet for å sikre en pålitelig og effektiv

<sup>30</sup> [Nasjonens hukommelse](#)

<sup>31</sup> [Arkivverkets årsrapport 2021](#)

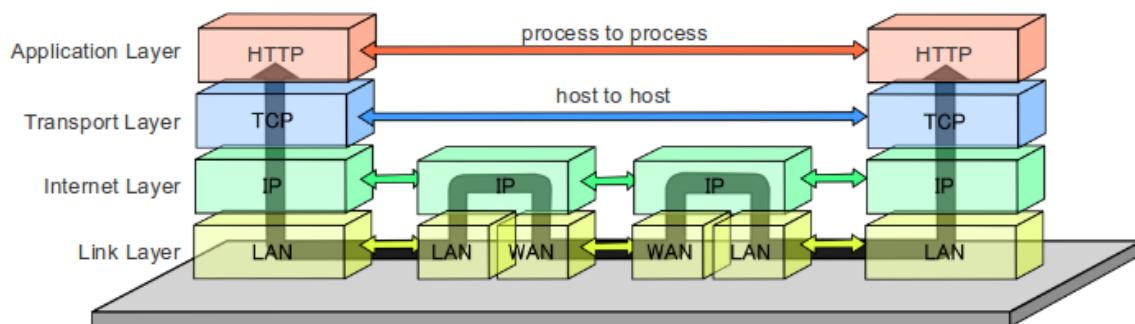
## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

dataoverføring gjennom internett. Dataene blir ikke sendt som en sammenhengende streng over en fast, fysisk forbindelse, men delt opp i pakker som kan ta forskjellige veier fra sender til mottaker hvor de blir satt sammen igjen i riktig rekkefølge. Dermed kan kommunikasjonen fungere selv om enkelte veier i nettverket er utilgjengelige.

Internett består hovedsakelig av kommunikasjonslinjer og rutere. Rutere er datamaskiner som fungerer som "veikryss" og bruker IP-protokollen til å rute datapakkene gjennom internett. Kommunikasjonslinjene, som for det meste er fiberkabler, forbinder rutere og følger standarder som PPP (*Point-to-Point Protocol*), HDLC (*High-Level Data Link Control*) eller *Frame Relay*. Internett benytter også trådløse forbindelser som satellittforbindelser, mikrobølgelink hvor vi møter på enda flere standarder for utveksling av data. Nedenfor ser vi en forenklet figur som viser hvordan en vanlig datautveksling foregår og hvilke standarder som er involvert.

**Data Flow of the Internet Protocol Suite**



[File:Data Flow of the Internet Protocol Suite.PNG - Wikimedia Commons](#)

Datamaskiner kobles til det nederste datalinklaget som et lokalnett (LAN) basert på [IEEE 802.3](#) (Ethernet) og [IEEE 802.11](#) (WiFi) standardene. WAN (Wide Area Network) standardene har vi beskrevet tidligere. Mobiltelefoner kan tilkobles til internett enten via WiFi eller mobilnettverket som er koblet til internett gjennom såkalte *gateway servere*. Det neste laget er nettverkslaget ([IP-protokollen](#)) som ruter pakkene fra sender til mottaker. Laget over der igjen heter transportlaget ([TCP-protokollen](#)) som sørger for at dataene kommer riktig fram. Dersom en pakke mangler eller inneholder feil hos mottageren, ber den om få tilsendt pakken på nytt. TCP-protokollen tillater dessuten at vi kan bruke datamaskinens IP-adresse ("telefonnummer") til flere logiske forbindelser samtidig. Vi kan eksempelvis se på video, samtidig som vi vil leser e-post, åpner websider eller chatter. På laget over transportlaget benytter programmene seg av protokollene i applikasjonslaget. *World Wide Web* (WWW) benytter [HTTP-protokollen](#). Det vil si, en *webserver*, eksempelvis IIS- eller Apache-programmet følger HTTP-protokollen når den sender websider til en nettsurfer, for eksempel Chrome- eller Safari-programmet. Vi bruker også HTTP-protokollen i Python når vi henter filer med `requests.get(url)` eller `pandas.read_json(url)`. Andre kjente protokoller i applikasjonslaget er FTP (filoverføring), SMTP (sende e-post), POP3 og IMAP (motta e-post) og DNS, som lar oss bruke navn i stedet for IP-adresser når vi kontakter andre datamaskiner. Så når vi skal lage et program som skal kommunisere over internett, behøver vi kun å forholde oss til en protokoll i applikasjonslaget eller skrive denne selv. De underliggende lagene tar seg automatisk av resten.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Standarder for sikring av ulike typer data

Vi har lært at vi blir stadig mer avhengige av dатateknologi, og mengden digitale data vokser eksponentielt. Derfor må vi stille større krav til sikringen av disse dataene og tilgjengeligheten til dem. Konfidensialitet, integritet og tilgjengelighet er tre sentrale prinsipper som standarder for sikring av data tar sikte på å opprettholde. Konfidensialitet handler om å sikre at informasjonen ikke blir kjent for uvedkommende, mens integritet garanterer at den ikke blir endret utilsiktet eller av uautoriserte personer. Videre er tilgjengelighet viktig for å sikre at informasjonen er tilgjengelig når det er behov for det.



Truslene mot datasikkerheten er mange og også stadig økende. Fra cyberkriminelle og skadelig programvare til interne uaktsomheter og naturkatastrofer, er det viktig å være oppmerksom på disse truslene og hvordan vi kan bidra til å bekjempe dem. Et eksempel på en slik trussel ble tydelig da Russland angrep Ukraina i 2022 ved å skyte krysserraketter mot Ukrainas statlige datasentre. Imidlertid hadde Ukraina tatt forholdsregler ved å "evakuere" sin digitale infrastruktur til nettskyen, der den ble driftet fra datasentre over hele Europa<sup>32</sup>.

#### Konfidensialitet

Førstelinjeforsvaret mot uautorisert tilgang til data er brukernavn og passord. Det finnes standarder for hvordan passordene lagres kryptert, for eksempel ved bruk av sterke kryptografiske *Hash*-funksjoner som **SHA-256** eller **SHA-3**. Disse standardene sikrer at passordene ikke kan gjenopprettes i sin opprinnelige form og reduserer risikoen for at de blir kompromittert ved et eventuelt sikkerhetsbrudd. I tillegg er det også standarder som angir hvordan utvekslingen av passord skal foregå mellom datamaskinen og serveren som godkjenner påloggingen. Eksempler på slike standarder er **Kerberos**, som brukes i en rekke nettverksmiljøer, inkludert Windows-domener, og **OAuth**, som er en protokoll for autentisering og autorisasjon på nettet. Disse standardene legger til rette for sikker overføring av passord eller autentiseringsdata mellom klienten og serveren, og bidrar til å beskytte mot uautorisert tilgang og passordrelaterte angrep. Videre brukes **LDAP**-protokollen til å aksessere og administrere katalogtjenester, som inneholder brukerinformasjon og annen relevant data for autentisering og autorisasjon. LDAP kan dermed integreres med disse standardene for å oppnå en helhetlig sikkerhetsløsning for håndtering av passord og brukerinformasjon. Dette var kun noen eksempler. Det finnes også andre tilsvarende standarder som PAM (*Pluggable Authentication Modules*) og NTLM (*NT LAN Manager*), som benyttes i ulike operativsystemer og nettverksmiljøer. I tillegg finnes det retningslinjer og beste praksiser for håndtering av passord og autentisering, som [NIST Special Publication 800-63B](#) og [OWASP Secure Coding Practices-Quick Reference Guide](#). Disse veiledningene gir anbefalinger og retningslinjer for hvordan man skal utforme og implementere sikre autentiseringssystemer.

<sup>32</sup> [Defending Ukraine: Early Lessons from the Cyber War](#)

Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Når det gjelder lagring av data, er det viktig å sikre at de blir lagret på en måte som beskytter dem mot uautorisert tilgang, selv om noen får fysisk tilgang til lagringsmediet. Her kommer *Advanced Encryption Standard (AES)* inn i bildet. AES er en krypteringsalgoritme som brukes til å sikre konfidensialiteten til dataene ved å konvertere dem til en uleselig form ved hjelp av en nøkkel. AES er en av de mest anerkjente og sikre krypteringsalgoritmene som brukes i dag.

Dessuten er data som overføres over Internett er sårbare for avlytting og det finnes standarder for hvordan dataene krypteres på de ulike nivåene i TCP/IP-modellen. WiFi-tilkoblinger kan krypteres med protokoller som **WPA**, **WPA-2** eller **WPA-3** for å sikre trådløs kommunikasjon. IP-protokollen kan beskyttes ved bruk av **VPN** (*Virtual Private Networks*), som oppretter en kryptert tunnel for sikker datatransport. På transportlagsnivået kan TCP-protokollen sikres med protokoller som **SSL** (*Secure Sockets Layer*) eller **TLS** (*Transport Layer Security*) for å oppnå sikker kommunikasjon mellom klient og server. Videre har protokollene i applikasjonslaget også fått sine sikre varianter. For eksempel har HTTP blitt erstattet med **HTTPS**, som legger til en krypteringsmekanisme ved hjelp av SSL/TLS for sikker overføring av data mellom nettlesere og nettsteder. FTP (*File Transfer Protocol*) kan også sikres med **FTPS**, en utvidelse som legger til SSL/TLS-kryptering for å beskytte filoverføringer. Telnet, som er en usikker protokoll for ekstern tilkobling til en annen datamaskin, er blitt erstattet med **SSH** (*Secure Shell*), som gir sikker kommunikasjon og autentisering. Ved lagring i nettskyen er det viktig å vurdere kryptering av data før de sendes, for eksempel ved bruk av AES.

For å oppnå best mulig sikkerhet bør det også implementere andre sikkerhetsmekanismer som totrinns pålogging, biometrisk godkjenning, brannmurer, antivirusprogrammer og sikkerhetsovervåking. Disse tiltakene bidrar til å styrke sikkerheten, selv om de ikke er standarder i seg selv.

#### *Integritet*

Dataintegritet handler om å sikre at data forblir nøyaktige, uendret og pålitelige gjennom hele levetiden. Blant de potensielle truslene mot dataintegriteten er uautoriserte endringer, feiltasting, skadelig programvare og sabotasje. Mens spionasje er en trussel rettet mot konfidensialiteten til data, der uautoriserte aktører søker å skaffe seg tilgang til sensitiv informasjon, er sabotasje en trussel som tar sikte på integriteten til data. Målet med sabotasje er å endre eller manipulere informasjonen for å påvirke beslutninger eller skape forstyrrelser og usikkerhet.

For å opprettholde dataintegriteten er det viktig å iverksette standarder og retningslinjer som sikrer at dataene forblir uendret og pålitelige. Vi har sett at sterke kryptografiske *Hash-funksjoner* som **SHA-256** og **SHA-3** kan generere unike *Hash-verdier* for dataene og sikre konfidensialitet, men de kan også brukes til å forsikre oss om at data ikke har blitt tuklet med. Digitale signaturer, implementert gjennom standarder som **RSA** og **DSA**, benytter asymmetrisk kryptografi for å sikre integriteten til dataene. *Hash Message Authentication Code (HMAC)* kombinerer en nøkkel og en *Hash-funksjon* for å generere en autentiseringskode, som sikrer både integriteten og autentiseringen av dataene.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

I tillegg til disse kryptografiske mekanismene finnes det anerkjente standarder og rammeverk som gir retningslinjer for å sikre dataintegritet på ulike nivåer i et system. Et eksempel på en slik standard er [ISO 27001](#), som gir generelle retningslinjer for informasjonssikkerhet og inkluderer også krav til dataintegritet. Det samme gjør [NIST SP 800-53 \(Security and Privacy Controls for Information Systems and Organizations\)](#): utgitt av *National Institute of Standards and Technology* i USA. Den gir retningslinjer og kontroller for informasjonssikkerhet, inkludert standarder for lagring, utveksling og sikring av data. Videre har vi bransjespesifikke standarder som *Payment Card Industry Data Security Standard (PCI DSS)* med spesifikke krav og anbefalinger for å sikre integriteten til data i henholdsvis betalingstransaksjoner.

Selv om implementering av standarder er viktig for å sikre dataintegriteten, er det også nødvendig å bruke beste praksiser og andre sikkerhetsmekanismer i tillegg. Som nevnt tidligere, er sikkerhetsmekanismer som totrinns pålogging, brannmurer, antivirusprogrammer og sikkerhetsovervåking avgjørende for å styrke sikkerheten på flere nivåer.

#### *Tilgjengelighet*

Noe av det første vi lærer eller erfarer når vi begynner å lagre data, er hvor viktig det er å ha sikkerhetskopier. Data kan gå tapt av ulike årsaker, som utilsiktet sletting, tekniske feil som diskkrasj eller overoppheving på grunn av feil i kjølesystemer, tyveri eller brann. Mulighetene er mange. **ISO 27001**-standarden, som tidligere nevnt i forbindelse med dataintegritet, inneholder også retningslinjer for sikkerhetskopiering av data som en viktig del av informasjonssikkerhetsprosessen. Den krever at organisasjoner etablerer dokumenterte rutiner for sikkerhetskopiering, inkludert risikovurdering, utvikling av sikkerhetskopieringspolitikk og prosedyrer, testing av gjenopprettingsprosesser, samt regelmessig overvåking og revisjon av sikkerhetskopieringssystemet.

Det finnes også andre grunner til at data kan bli utilgjengelige. DDOS-angrep og skadeware utgjør betydelige trusler mot tilgjengeligheten til data. Et DDOS-angrep (*Distributed Denial of Service*) er en ondsinnet handling der en rekke datamaskiner blir mobilisert til å oversvømme en tjeneste eller nettverk med trafikk, noe som resulterer i at tjenesten blir utilgjengelig for legitime brukere. Dette kan føre til nedetid og forstyrrelser i tilgangen til data. Skadeware er en annen risiko som kan forhindre tilgang til data. Skadeware, som virus, *ransomware* eller trojanere, kan infiltrere systemer og forårsake skade, inkludert kryptering eller sletting av data. Dette kan resultere i tap av tilgang til viktige filer og informasjon. Selv om det ikke finnes en spesifikk standard for beskyttelse mot DDOS-angrep og skadeware, inneholder ISO 27001 flere kontroller og tiltak som kan bidra til å styrke sikkerheten mot disse truslene, for eksempel brannmurer og sikkerhetsovervåkning.

#### *Helhetlig tilnærming*

I korte trekk er standarder for sikring av ulike typer data nødvendig for å opprettholde konfidensialitet, integritet og tilgjengelighet av data. Men standarder alene er ikke tilstrekkelig i denne sammenhengen. I tillegg til standardene er etablering av retningslinjer og beste praksis avgjørende for å bedre beskytte våre systemer og nettverk mot trusler som skadeware, hacking og uautorisert tilgang. Brannmurer, antivirusprogramvare, sikkerhetsovervåkning og andre sikkerhetsverktøy utgjør også viktige komponenter i vår

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

samlede sikkerhetsstrategi. I tillegg kommer opplæring av brukere og jevnlig oppdatering av systemene våre for å begrense risikoen for uautorisert tilgang og skadelig aktivitet. Ved å kombinere standarder, retningslinjer, verktøy, opplæring og oppdateringer, oppnår vi en helhetlig tilnærming som styrker vår datasikkerhet og gir oss bedre mulighet til å beskytte dataene våre.

### Ta med minst dette

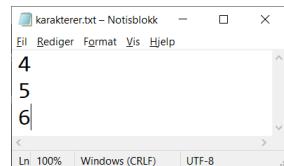
For å håndtere data riktig, må vi forstå og bruke standarder for lagring, utveksling og sikring av data. Disse standardene sikrer at dataene lagres på en organisert måte, overføres pålitelig mellom systemer, og beskyttes mot uautorisert tilgang og endringer. Det handler om konfidensialitet, integritet og tilgjengelighet av data. Standarder fører oss et godt stykke på veien, men de alene er ikke nok. Vi må også ha retningslinjer, beste praksis, sikkerhetsverktøy og opplæring for å beskytte systemene våre effektivt.

### Øvingsoppgaver

#### 4.5.1

I denne øvingen skal vi se hvordan SHA-koden kan avsløre hvorvidt innholdet i en fil har blitt tuklet med.

1. Åpne PowerShell ved å søke etter **PowerShell** i startmenyen eller ved å trykke på **Win+X** og velge **Windows PowerShell**.
2. Naviger til mappen for skrivebordet med **cd**-kommandoen (*Change Directory*):  
`cd $home\Desktop`
3. Sjekker at vi står i riktig mappe med **Get-Location**-kommandoen:  
`gl`
4. Opprett en ny fil ved hjelp av **New-Item**-kommandoen:  
`New-Item -ItemType File -Name "karakterer.txt"`
5. Åpne filen i Notepad, og legg inn karakterene 4,5 og 6.  
`notepad karakterer.txt`
6. Sjekker at vi har riktig innhold med **Get-Content**-kommandoen:  
`gc karakterer.txt`
7. Bruk **Get-FileHash**-kommandoen til å beregne SHA-koden. Skriv følgende:  
`Get-FileHash -Path "karakterer.txt" -Algorithm SHA256`
8. Ta vare på *Hash*-koden i et annet **.txt**-dokument.
9. Endre innholdet i **karakterer.txt** til **5 5 og 6**.
10. Bruk **Get-File-Hash**-kommandoen på nytt.
11. Kopier denne *Hash*-koden under den forrige i den andre **.txt**-dokumentet.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

12. Søk etter den første *Hash*-koden i dokumentet. Hvis vi ikke finner noen flere like *Hash*-koder, er den nye *Hash*-koden forskjellig fra den første, og dokumentet må ha blitt endret.

### 4.5.2

I denne oppgaven skal vi lære hvordan vi kan være sikre på at dataene virkelig er slettet.

1. Skriv en kort innledning som forklarer hvorfor sikker datasletting er viktig og hvilke risikoer som kan oppstå hvis dataene ikke blir slettet ordentlig.
2. Formater den utdelte minnepinnen.
3. Opprett to tekstfiler med enkelt innhold.
4. Slett den første med **Delete**-tasten og den andre med **Shift-Delete**-tastekombinasjonen.
5. Sjekk papirkurven og prøv å gjenopprett filene.
6. Last ned **recuva**-programmet fra <https://www.ccleaner.com/recuva/download>
7. Installer programmet og prøv å gjenopprette filene.
8. Forklar hva som har skjedd: Diskuter resultatene og hvorfor filene kunne gjenopprettes etter bruk av vanlig sletting og papirkurven.
9. Last ned **eraser**-programmet fra <https://eraser.heidi.ie/download/>
10. Opprette en tekstfil med enkelt innhold.
11. Høyre-klikk på filen og velg **Eraser** fra nedtrekksmenyen.
12. Prøv å gjenopprette filen.
13. Forklar hva som har skjedd: Diskuter resultatene og hvorfor filen ikke kunne gjenopprettes etter bruk av Eraser-programmet.
14. Oppsummer erfaringene og reflekter over betydningen av sikker datasletting. Diskuter hvorfor det er viktig å ta vare på personlige data og hvordan riktig sletting kan bidra til personvern og datasikkerhet.

### 4.5.3

Vi har mottatt en CSV-fil med data som skal brukes av et annet program. I midlertid kan dette programmet kun lese JSON-filer. Oppgaven er å skrive et Python-program som konverterer CSV-filen til en JSON-fil.

1. Forklar kort forskjellen mellom CSV og JSON filformater og hvorfor konvertering er nødvendig i dette tilfellet.
2. Gjør deg kjent med strukturen og innholdet i den vedlagte CSV-filen, [4\\_5\\_personer.csv](#).
3. Skriv et Python-program som leser innholdet i CSV-filen, konverterer det til JSON-format og lagrer innholdet i JSON-filen [4\\_5\\_personer.json](#)
4. Test programmet ved å kjøre det med **personer.csv** som inndata.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

5. Verifiser at programmet produserer en gyldig JSON-fil med riktig struktur og data fra CSV-filen.
6. Kjør vedlagte program [4\\_5\\_les\\_json.py](#) og kontroller utskriften blir som vist nedenfor.

```
fornavn etternavn    yrke   alder
0      Per     Hansen    Selger    43
1    Yngve     Lie     Maler    25
2      Lise   Hussain     Lege    45
3  Charlotte  Høyheim  Ingeniør    33

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   fornavn    4 non-null      object  
 1   etternavn  4 non-null      object  
 2   yrke       4 non-null      object  
 3   alder      4 non-null      int64  
dtypes: int64(1), object(3)
memory usage: 256.0+ bytes
None
```

7. Forklar stegene du har fulgt for å konvertere CSV til JSON ved hjelp av Python og diskuter eventuelle utfordringer du møtte underveis.
8. Diskuter hvorfor standardisering av datautvekslingsformater som CSV og JSON er viktig for effektiv og pålitelig datautveksling mellom forskjellige programmer.

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 4.6 Flere verktøy for samarbeid.

#### Bolkens innhold

Innledning .....	285
Live Share.....	285
GitHub.....	286
Trello.....	288
Slack.....	288
Integrasjoner .....	289
Ta med minst dette.....	290
Øvingsoppgaver.....	290

#### Kompetansemål:

- velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre

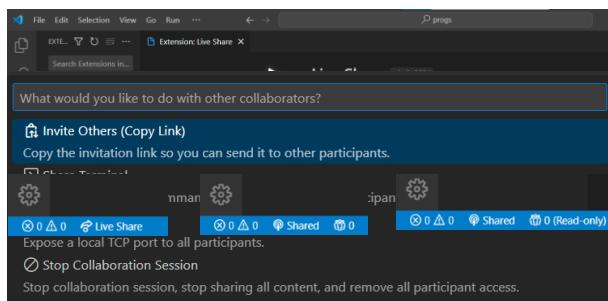
#### Innledning

I et systemutviklingsprosjekt er det viktig å jobbe effektivt og samarbeide godt for å oppnå best mulig resultat. Vi har allerede lært om noen grunnleggende verktøy og teknikker for samarbeid, slik som å velge gode variabel- og funksjonsnavn, bruk av kommentarer i koden og API-dokumentasjon, samt bruk av UML-klassediagram, pseudokode, flytskjema og *wireframes*. Nå skal vi se på andre verktøy som vil hjelpe oss å samarbeide enda bedre. I denne bolken vil vi lære om **Live Share** i VS Code, **GitHub**, **Trello** og **Slack**. *Live Share* gir oss muligheten til å samarbeide og kode i sanntid. *GitHub* lar oss samarbeide om en felles kodebase (*shared repository*) fra Git i VS Code. *Trello* er et prosjektstyringsverktøy som bruker visuelle tavler og kort for å organisere og administrere oppgaver og prosjekter, mens *Slack* er en gratis programvare som gir oss muligheten til å kommunisere via taleanrop, videosamtaler, tekstmeldinger, deling av media og filer i chatten, og kan integreres med andre verktøy som *GitHub* og *Trello*.



#### Live Share

*Live Share* er en VS Code-utvidelse som lar oss dele skjermen med andre. Det er en "kom-i-gang"-veiledning på velkomstsiden hvor vi også finner en [lenke](#) til mer dokumentasjon hos Microsoft. Når vi har installert *Live Share*-utvidelsen i VS Code, er det enkelt å starte en *Live Share*-økt og



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

invitere andre til å delta. Klikker vi på *Live Share*-ikonet nede til venstre i vinduet, får vi opp et vindu hvor vi kan velge om økten skal kun ha lesertilgang eller ikke. Lenken kan vi også dele i *Slack* eller *Trello* når vi har lært å bruke disse verktøyene. Ikonet endrer seg fra *Live Share* til *Shared*, og klikker vi på det nå, får vi opp en meny med fire valg. Vi kan kopiere lenken på nytt, dele terminal, dele en tjener eller avslutte økten.

*Live Share* er generelt trygt å bruke, siden det er satt opp flere sikkerhetstiltak for å beskytte både koden og datamaskinen vår mens vi deler skjermen med andre. Likevel er det viktig å være forsiktig og oppmerksom på sikkerhetsrisikoer når vi deler koden med andre, spesielt i profesjonelle sammenhenger på internett.

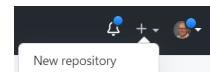
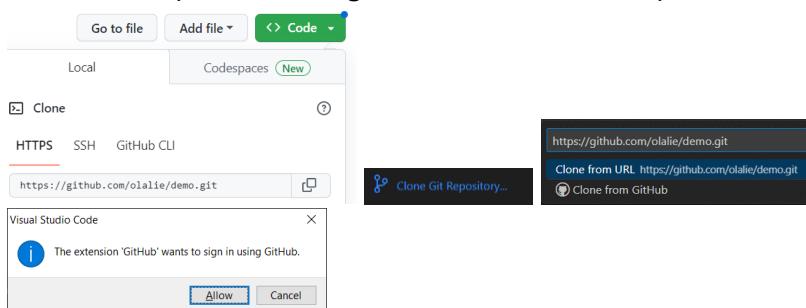
### GitHub

I 2.3 Versjonskontroll med Git lærte vi å bruke Git på vår egen datamaskin. Nå vil vi bruke Git på en annen måte som gir oss muligheten til å samarbeide med andre ved å jobbe med en kopi av koden som ligger i skyen på GitHub. Vi kunne ha brukt en delt mappe for å lagre koden, men da ville ikke fått tilgang til nyttige samarbeidsverktøy i GitHub, som *pull requests* (forespørsel til andre om å *merge* vår *branch* med *main*), *issues* (følge opp problemer), *code reviews* (andre kommenterer koden vår) og *project boards* (prosjektstyringsverktøy). Selv om GitHub er den største Git-baserte skytjenesten, finnes flere andre tilbydere som GitLab, Bitbucket, SourceForge og Azure DevOps.

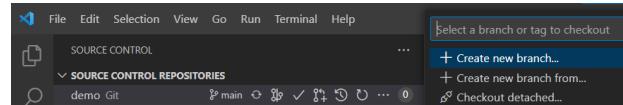
Her er en mulig rutine for hvordan vi kan jobbe med VS Code og GitHub.

#### Kun én gang:

1. En GitHub-bruker oppretter et tomt *repo* og legger til de andre GitHub-brukerne som skal delta i prosjektet.
2. Alle kloner *repo*-et til sin egen datamaskin (via Help, Welcome i VS Code).

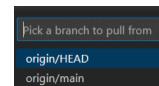


3. Hvert medlem lager sin arbeidsgren (*branch*) for endringene/tilleggene sine. Klikk på *main* og velg *Create new branch...*



#### For hver arbeidsøkt:

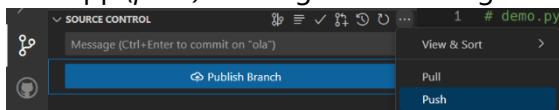
4. Stå i arbeidsgrenen, hent (*pull*) endringer fra hovedgrenen (*main*) i GitHub og løs eventuelle konflikter (..., Pull, Push, Pull from...).
5. Husk å fullføre (*commit*) eventuelle endringer.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

6. Skriv kode.
7. Legg til (*add*) og fullfør (*commit*) endringene med en beskrivende melding.
8. Last opp (*push*) endringene i arbeidsgrenen til GitHub.

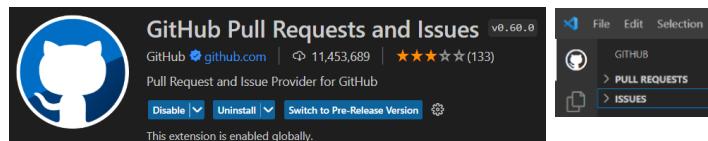


#### Når vi er ferdig:

1. Opprett en forespørsel (*pull request*) om å slå sammen (*merge*) arbeidsgrenen med hovedgrenen på GitHub. Beskriv endringene og be andre om å gjennomgå koden (*code review*).
2. Løs eventuelle problemer som oppstår under gjennomgangen.
3. De som til slutt godkjenner koden, kan flette den sammen (*merge*) med hovedgrenen.
4. Slett arbeidsgrenen i GitHub og lokalt (Endringen finnes fremdeles i historikken).

Disse stegene kan vi også bruke i andre faser av utviklingen, som når vi skriver tester, fikser feil eller lager dokumentasjonen.

Når vi jobber med *pull requests* og *code reviews*, kan vi gjøre direkte på GitHubs websider eller via utvidelsen *GitHub Pull Requests and Issues* i VS



Code. Det kan være lurt å lære det først på GitHubs websider.

Hvis et medlem *pusher* eller *syncer* endringer uten å være på en *branch*, vil endringene gå direkte inn i *main* på GitHub, uten *code review* og *pull request*. Dette kan være uheldig i et større prosjekt, så da kan den som eier *repo*-et, klikke på *Protect this branch* og kreve at endringer gjøres gjennom *pull requests* og *code reviews*. På den andre siden kan et slikt krav medføre ineffektivitet i et lite prosjekt.

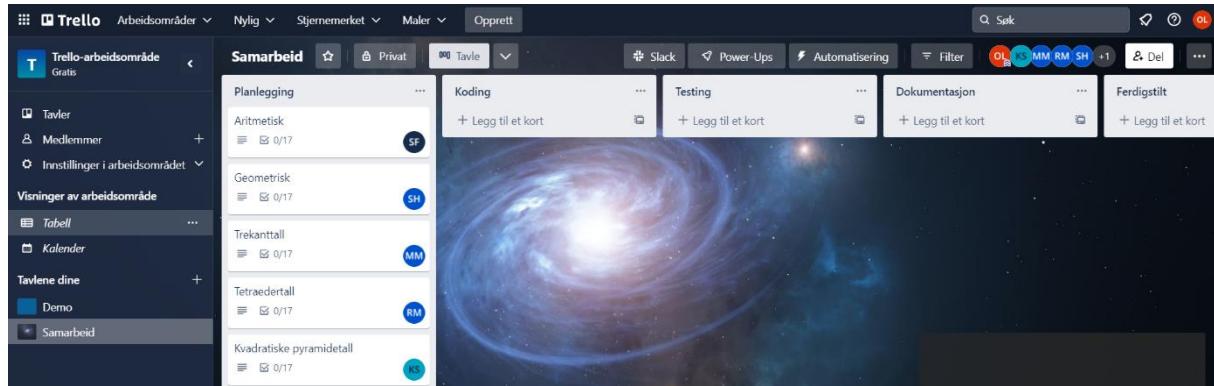
Standardoppsettet for GitHub sender e-postvarsler til teammedlemmer om viktige hendelser. Disse varslene kan imidlertid slås av eller sendes til andre kommunikasjonsverktøy, som for eksempel *Slack*.

I øvingsoppgavene vil lære og bruke disse grunnleggende funksjonene i GitHub, og det vil gi oss et godt utgangspunkt for å lære mer siden. I profesjonelle prosjekter er det mange andre ting å vurdere, som for eksempel navnkonvensjoner for *brancher*, *commits* og *tags*, samt mer avanserte teknikker som kontinuerlig integrering (*CI, Continuous Integration*) og kontinuerlig utrulling/levering (*CD, Continuous Delivery/Deployment*). Å lære de mer avanserte funksjonene vil kreve mer tid enn vi har til rådighet, og vi mener det ikke er nødvendig for å oppfylle læreplanens kompetanse mål om å "velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre".

## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

### Trello

Det er begrenset hvor mange detaljer vi klarer å huske, og når det blir for mange av dem, kan det være lurt å skrive dem opp. Slik er det også i systemutvikling når vi trenger prosjektstyringsverktøy til å holde orden på oppgaver og aktiviteter i et prosjekt. Slike prosjekter kan koste fra noen hundre tusen til flere milliarder kroner. Jo større prosjekter, desto mer avanserte prosjektstyringsverktøy trenger vi, og det er mange å velge mellom. For små prosjekter kan verktøy som *Trello* eller *Basecamp* være tilstrekkelig, mens for større prosjekter kan verktøy som *Asana*, *Smartsheet*, *Oracle Primavera* eller *SAP PS* være mer passende.

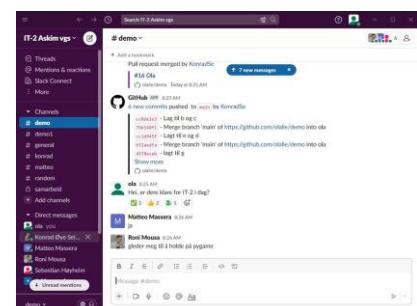


Vi skal bruke *Trello*, som er webbasert og hjelper oss å organisere oppgaver på en effektiv måte. Med *Trello* kan vi dele opp prosjektet i mindre deler som kalles kort og organisere dem i lister. *Trello* er basert på kanban-metoden, som opprinnelig stammer fra japansk industriproduksjon. Målet var å optimalisere flyten av arbeidet fra stasjon til stasjon i produksjonslinjen. I et systemutviklingsprosjekt kan det være naturlig å tenke på stasjonene (listene i *Trello*) som faser og navngi dem med ord som for eksempel planlegging, koding, testing, dokumentasjon og ferdigstilt. Når vi er ferdige med en fase, flytter vi *Trello*-kortet til neste liste (fase/stasjon). Vi kan også legge til kommentarer og filer, tildele ansvar og sette frister. Dette gir alle full oversikt over hva som må gjøres, hvem som gjør det, og når det skal være ferdig.

*Trello* er intuitivt og lett å lære. Det kan også integreres med *Slack* for å sende automatisk beskjeder om viktige hendelser til en *Slack*-kanal. I IT-2 bruker vi *Trello* i et systemutviklingsprosjekt, men det kan også brukes til mange andre formål, som for eksempel personlige gjøremål, skoleprosjekter og mye mer.

### Slack

Stadig flere av oss bruker samarbeids- og kommunikasjonsverktøy, enten det er i arbeidslivet eller i skolesammenheng. Med økt bruk av hjemmekontor, fjernarbeid, outsourcing og globalisering i arbeidslivet, øker også behovet for verktøy som kan hjelpe oss med å samarbeide på tvers av geografiske avstander og tidssoner. I skolen har vi sett økt bruk av fjernundervisning og hjemmeskole under koronapandemien. Det finnes flere



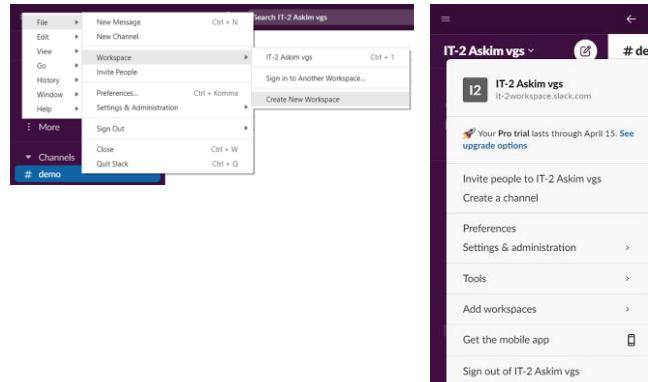
## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

samarbeids- og kommunikasjonsverktøy å velge blant, som Teams, Zoom, Discord og Slack. Disse verktøyene har funksjoner som *chat*<sup>33</sup>, direktemeldinger, kanaler, (audio- og) videosamtaler, fil- og skermdeling samt integrasjoner.

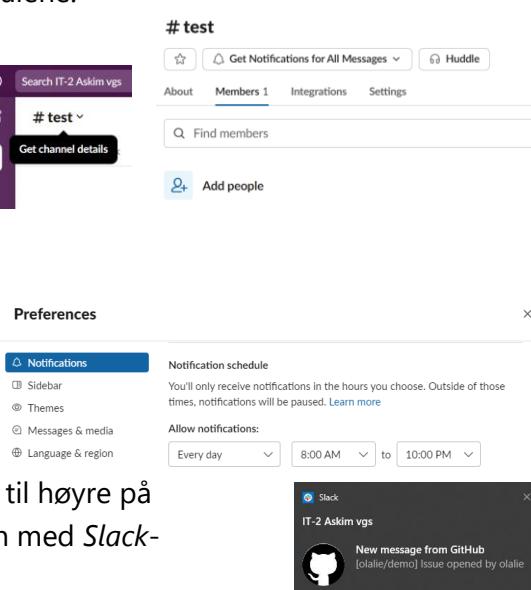
Systemutviklingsprosjekter kan være komplekse og krevende, og det er ofte behov for spesialiserte samarbeidsverktøy for å håndtere prosjektene på en effektiv måte. Disse verktøyene kan hjelpe oss med å dele kode, diskutere tekniske problemer, planlegge og organisere oppgaver, og samarbeide fra hvor som helst, når som helst.

Vi har valgt å introdusere *Slack* fordi det [dominerer hos norske utviklere](#). Ifølge Kode 24 brukte 63% av norske utviklere *Slack* og bare 16% Teams, som kom på en god annenplass, i november 2019. Det kan være fordi *Slack* er enkelt å sette opp, enkelt å integrere med andre programmer og enkelt å bruke. I tillegg tilbyr verktøyet *chat*, direktemeldinger, kanaler og videosamtaler, noe som dekker vårt behov for effektiv kommunikasjon.

Arbeidsområder kan vi opprette fra hovedmenyen opp til venstre. De kan vi konfigurere ved å klikke på de respektive navnene som dukker opp. Under *Settings & Administration* kan vi legge til medlemmer.



Kanaler oppretter vi ved å klikke på *Channels* og velge *Create*, og når vi har åpnet en kanal, kan vi klikke på kanalnavnet (*Get channel details*) i vinduet for å konfigurere den. Vi kan invitere medlemmer til både arbeidsområdet og kanalene.



### Integrasjoner

Både *GitHub* og *Trello* kan integreres med *Slack*, og kommunikasjonen kan gå begge veier. Vi vil begrense oss til meldinger om viktige hendelser fra *GitHub* og *Trello* til en kanal i *Slack*. Meldingen kan også dukke opp som varsler i små vinduer nede til høyre på skjermen hvis vi ønsker det. Det krever at nettsleseren med *Slack*-

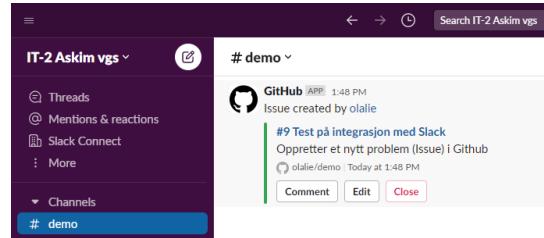
<sup>33</sup> Nettprat

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

arkfanen er åpen, så det er bedre å starte *Slack*-appen, som kan være minimert. Meldingene blir postet i *Slack*-kanalen selv om vi velger å slå av varslene. *Preferences* får fram ved å klikke på ikonet vårt opp til høyre i vinduet.

For å sette opp en integrasjon mellom Slack og Trello, klikker vi på *Power-Ups*-knappen i Trello og legger til *Slack*. Deretter kan vi klikke på *Slack*-knappen å angi arbeidsområde, kanal og hvilke handlinger vi ønsker å sende. I Slack må vi klikke på kanalen, velge Integrations og Add Apps.



### Ta med minst dette

Vi har utforsket fire nye verktøy som styrker samarbeidet i systemutviklingsprosjekter. **Live Share i VS Code** muliggjør skjermdeling, slik at flere kan kode i samme vindu i samtid. Tidligere har vi brukt Git til versjonskontroll av vår egen kode. **GitHub** ivaretar denne versjonskontrollen, men lagrer koden i skyen. Dermed kan vi samarbeide ved å laste ned oppdaterte lokale kopier av koden og laste opp endringer tilbake til en felles kodebase. **Trello** gir oss visuell prosjektstyring ved hjelp av kort med aktiviteter og lister med kort. Aktivitetene kan hukes av etter hvert som de blir utført, og kortene kan flyttes til en ny liste når vi kommer til en ny fase i prosjektet. Med **Slack** kan vi kommunisere ved hjelp av direkte meldinger og kanaler. GitHub og Trello kan integreres med Slack slik at vi mottar varsler i Slack når noe blir endret i GitHub eller Trello. Alle i prosjektet bør derfor være oppdatert til enhver tid og kunne reagere raskt og effektivt.

### Øvingsoppgaver

Samarbeidsprosjekt: Tallrekker

Dette prosjektet forutsetter følgende for hver prosjektgruppe

1. Det er opprettet et *repo samarbeid* på GitHub hvor elevene har tilgang
2. *Repo*-et har en tom **tallrekker.py**, **test\_tallrekker.py** og en **docs** mappe.
3. *Repo*-et har en **README.md** med beskrivelse av prosjektet:

### Prosjekt: Samarbeid

#### Formål

Formålet med prosjektet er å lære å bruke verktøyene Live Share og GitHub, Trello og Slack ved å samarbeide i et lite utviklingsprosjekt.

#### Prosjektbeskrivelse

Vi skal lage en Python-modul kalt **tallrekker.py** som inneholder fem funksjoner. Disse funksjonene skal returnere de første n elementene i følgende tallrekker:

1. aritmetisk(n, a, d) som tar inn antall elementer n, startverdi a og differansen d, og returnerer de n første tallene i en aritmetisk rekke.
2. geometrisk(n, a, r) som tar inn antall elementer n, startverdi a og forholdstallet r, og returnerer de n første tallene i en geometrisk rekke.
3. trekanttall(n) som tar inn antall elementer n, og returnerer de n første trekanttallene.
4. tetraedertall(n) som tar inn antall elementer n, og returnerer de n første pyramidettallene hvor grunnflaten er en likesidet trekant.
5. kvadratiske\_pyramidettall(n): Funksjonen tar inn inn antall elementer (n), og returnere de n første pyramidettallene hvor grunnflaten er et kvarat.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Vurderingsseksemplar

### Arbeidsfordeling

1. aritmetisk(n, a, d) -> Hasanah
2. geometrisk(n, a, r) -> Sebastian
3. trekanttall(n) -> Matteo
4. tetraedertall(n) -> Roni
5. kvadratiske\_pyramidetall(n) -> Konrad

### Tidsplan

- Alt arbeid skal utføres i uke 12, 2023.
- Digitalt møte i Slack mandag 20.03.23 kl 15:15
- Digitalt møte i Slack onsdag 20.03.23 kl 09:15
- Det skal planlegges et digital møte i slutten av uken.

### Kommunikasjon

- Vi bruker Slack som vår primære kommunikasjonskanal og har opprettet en egen Slack-kanal for prosjektet hvor vi kan diskutere fremdrift, dele filer og gi tilbakemeldinger.
- Vi vil ha jevnlige videomøter i Slack for å diskutere fremdrift og eventuelle problemer, og bruker Live Share under møtene slik at vi kan se og endre kode i sanntid.
- Vi vil oppdatere Trello-kortene med fremdriftsrapporter og eventuelle endringer i oppgavene våre. Det er en viktig kilde til informasjon for alle teammedlemmer.
- Hvis noen av oss har spørsmål eller trenger hjelp, skal vi ikke nøle med å ta kontakt med hverandre på Slack eller under et møte. Vi ønsker å oppmuntre til åpen kommunikasjon og samarbeid.

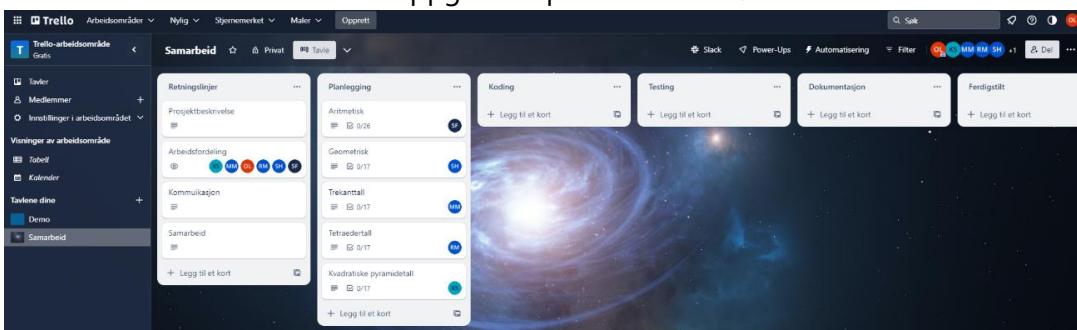
### Samarbeid

Vi bruker følgende verktøy:

- **Live Share** til å dele skjermbildet i sanntid for å å samarbeide om kodeutvikling og feilsøking .
- **GitLab** for en felles kodebase hvor vi legge ut koden vår og jobber sammen om å gjøre endringer og gi tilbakemeldinger.
- **Trello**: hvor vi organiserer oppgavene holder oversikt over fremdriften i prosjektet.
- **Slack** til å kommunisere i en egen kanal for prosjektet til til å koordinere arbeidet vårt.

Vi har satt opp automatiske varslinger i Slack for å holde teamet oppdatert på viktige endringer i GitLab og Trello, og vil dessuten ha jevnlige møter for å diskutere fremdriften og eventuelle problemer vi måtte møte på.

4. Det er opprettet en **Trello-tavle Samarbeid**, hvor elevene er lagt til og fått ansvar for hver sitt kort (Aritmetisk, Geometrisk, Trekanttall, Tetraedertall og Kvadratiske pyramidetall).
5. Hvert kort er klargjort med beskrivelse av oppgaven og sjekklisten for hver av fasene, Planlegging, Koding, Testing og Dokumentasjon. Listen til venstre, Retningslinjer, inneholder kun informasjon på kort som ikke skal flyttes (Prosjektbeskrivelse, Arbeidsfordeling, Kommunikasjon og Samarbeid). Listen til høyre, Ferdigstilt, er hvor de andre kortene havner når alle oppgavene på kortet er utført



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

6. Hvert kort er klargjort med fire sjekklisteler, Planlegging, Koding, Testing og Dokumentasjon. Når alle punktene i en sjekkliste er utført, flyttes kortet til neste liste.

**Aritmetisk**  
i listen [Planlegging](#)

Medlemmer Varsler SF + Overvåk

**Beskrivelse** [Rediger](#)

Lag en funksjon **aritmetisk(n, a, d)** som tar inn antall elementer **n**, startverdi **a** og differansen **d**, og returnerer de **n** første tallene i en aritmetisk rekke.

Bruk **fornavnet** ditt når du navngir *branchen*. Ha med **navnet på funksjonen** i *commit*-meldingen.

**Planlegging** Slett

0% Slett

- Klonet reporet
- Laget arbeidsgren (branch)
- Laget UML diagrammer i /docs mappen
- Lagt til (added) diagrammene, fullført (committed) og lastet opp arbeidsgrenen (pushed)
- Bedt om kodegjennomgang og forespørsl om å slå sammen arbeidsgrenen med hovedgrenen på GitHub (pull request)
- Løst eventuelle problemer og slått sammen arbeidsgrenen med hovedgrenen (merged)
- Slettet arbeidsgrenen på GitHub og lokalt

**Koding** Slett

0% Slett

- Laget ny arbeidsgren (branch) og hentet (pulled) hovedgrenen fra GitHub
- Kodet i tallrekker.py
- Lagt til (added) koden, fullført (committed) og lastet opp arbeidsgrenen (pushed).
- Bedt om kodegjennomgang og forespørsl om å slå sammen arbeidsgrenen med hovedgrenen på GitHub (pull request).
- Løst eventuelle problemer og slått sammen arbeidsgrenen med hovedgrenen (merged)
- Slettet arbeidsgrenen på GitHub og lokalt

**Testing** Slett

0% Slett

- Laget ny arbeidsgren (branch) og hentet (pulled) hovedgrenen fra GitHub
- Kodet og kjør vellykkede enhetstester i test\_tallrekker.py.
- Laget testdokumentasjon i /docs mappen.
- Lagt til (added) testkoden og dokumentasjonen, fullført (committed) og lastet opp arbeidsgrenen (pushed).
- Bedt om kodegjennomgang og forespørsl om å slå sammen arbeidsgrenen med hovedgrenen på GitHub (pull request).
- Løst eventuelle problemer og slått sammen arbeidsgrenen med hovedgrenen (merged)
- Slettet arbeidsgrenen på GitHub og lokalt

**Dokumentasjon** Slett

0% Slett

- Laget ny arbeidsgren (branch) og hentet (pulled) hovedgrenen fra GitHub
- laget API-dokumentasjon i /docs mappen
- Lagt til (added) api-dokumentasjonen, fullført (committed) og lastet opp arbeidsgrenen (pushed).  
Du har redigert dette feltet, men det er ikke lagret. [Se endringer som er gjort](#) • [Avvis](#)
- Bedt om kodegjennomgang og forespørsl om å slå sammen arbeidsgrenen med hovedgrenen på GitHub (pull request).
- Løst eventuelle problemer og slått sammen arbeidsgrenen med hovedgrenen (merged)
- Slettet arbeidsgrenen på GitHub og lokalt

Vurderingsksemplar

© TIP AS, 2024

Bolk 4.6

Side 292 av 382

## 4.7 Mer om OOM

### Bolkens innhold

Utvikle selv eller kjøpe ferdig?.....	293
Kravspesifikasjon.....	293
Bruksmønstre .....	294
Bruksmønsterdiagram.....	295
Klassediagram.....	296
Sekvensdiagram.....	296
Fra OOM til OOP.....	297
Ta med minst dette.....	299
Øvingsoppgaver.....	299

### Kompetansemål:

- anvende objektorientert modellering til å beskrive et programs struktur
- utforske og vurdere alternative løsninger for design og implementering av et program
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode

### Utvikle selv eller kjøpe ferdig?

Når vi trenger et datasystem, er det lurt å først sjekke om det finnes et ferdiglaget system som kan dekke våre behov. Selv om det kan hende at vi ikke får alt vi ønsker oss, er det som regel billigere enn å utvikle et skreddersydd system. Likevel kan en bedrift velge å utvikle et eget system som kan gi konkurransefortrinn og økt lønnsomhet på sikt.

### Kravspesifikasjon



Før vi starter utviklingen, må vi utarbeide en kravspesifikasjon som dokumenterer alle kravene systemet må oppfylle. Det vil være funksjonelle krav, men også krav til ytelse, brukervennlighet, pålitelighet, implementering, vedlikehold, juridiske krav, med mer. Kravspesifikasjonen vil trolig endre seg underveis, og derfor kan en smidig utviklingsmodell være en god løsning. Utarbeidelsen av en kravspesifikasjon kan være en omfattende oppgave, og mange bedrifter velger å hyre inn konsulenter for å hjelpe til med dette. Kravspesifikasjonen inngår i kontrakten om levering av systemet, sammen med andre forhold som pris, leveringstid, aksepttesting, eierskap, garanti, ansvar, taushetsplikt, bestillingsendringer, kontraktbrudd, osv. Utviklerne må også ha kompetanse i kravspesifikasjoner for å kunne verifisere at kravene er realistiske og gjennomførbare,

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

identifisere eventuelle manglende krav, lage sin egen kravspesifikasjon for modellering, koding, testing, vedlikehold, kommunikasjon med kunden, med mer.

Oppgavene vi arbeider med i IT-2 fungerer som kravspesifikasjoner for å løse de gitte problemene. Selv om vi ikke kommer til å utvikle kravspesifikasjonene selv, er det likevel viktig å forstå hva som inngår i en komplett kravspesifikasjon og hvor mye informasjon som er nødvendig for å utvikle gode systemer.

Når vi skal beskrive de funksjonelle kravene til et system, kan vi blant annet anvende bruksmønstre. Disse er hovedsakelig tekstdokumenter som beskriver samspillet mellom brukerne og systemet. UML spesifiserer også bruksmønsterdiagrammer som gir en oversiktlig visualisering av systemet. Etter at bruksmønstre og klassediagrammer er på plass, kan neste steg være å lage UML sekvensdiagrammer som viser hvordan objekter samhandler med hverandre og i hvilken rekkefølge. Dette hjelper oss med å identifisere hvilke metoder som må implementeres i objektene, og gir en oversikt over hvordan objektene i systemet samhandler med hverandre og med brukeren. Mens klassediagrammene er som et bilde av systemet, den statiske delen, er sekvensdiagrammene er som en film av systemet, når det er i bruk, den dynamiske delen. Det er vanlig å lage klassediagrammene først, men vi går frem og tilbake og justerer både klassediagrammene og sekvensdiagrammene etter hvert som vi lærer stadig mer om systemet.

Avslutningsvis kan vi si at planleggingsfasen består av både analyse og design. Kravspesifikasjon og bruksmønstre hører typisk til analysefasen, og er viktige verktøy for å identifisere hva systemet skal gjøre og hvordan det skal brukes. Klassediagrammer og sekvensdiagrammer er derimot en del av designfasen og bidrar til å visualisere hvordan systemet skal implementeres og hvordan objektene i systemet samhandler med hverandre. I tillegg er det verdt å nevne at i smidig systemutvikling går vi gjennom flere iterasjoner av planlegging, koding og testing. Dette betyr at vi ikke nødvendigvis fullfører all planlegging før vi begynner å kode, men heller jobber i kortere iterasjoner for å sikre at vi hele tiden har en god forståelse av hva som skal utvikles og hvordan det skal implementeres. På denne måten kan vi kontinuerlig tilpasse oss endringer og behov som oppstår underveis i utviklingsprosessen.

#### Bruksmønstre

For å utvikle gode systemer må vi forstå brukernes behov. Det krever et felles språk som begge parter forstår. Da er bruksmønstre (*Use Cases*) et nyttig verktøy. De beskriver i klartekst hvordan et system skal brukes for å oppnå det vi ønsker. Bruksmønstre kan variere fra [formelle og detaljerte dokumenter](#) til enklere skjemaer som dette:

- |                        |  |
|------------------------|--|
| <b>Tittel</b>          | : Forteller om handlingen og hva som skal oppnås               |
| <b>Aktører</b>         | : Hvem bruker systemet, kan også være andre systemer           |
| <b>Hovedflyt</b>       | : Gir en trinnvis fremstilling av hendelsesforløpet            |
| <b>Alternativ flyt</b> | : Beskriver hendelsesforløpet når ting ikke går som forventet. |

## Smidig IT-2

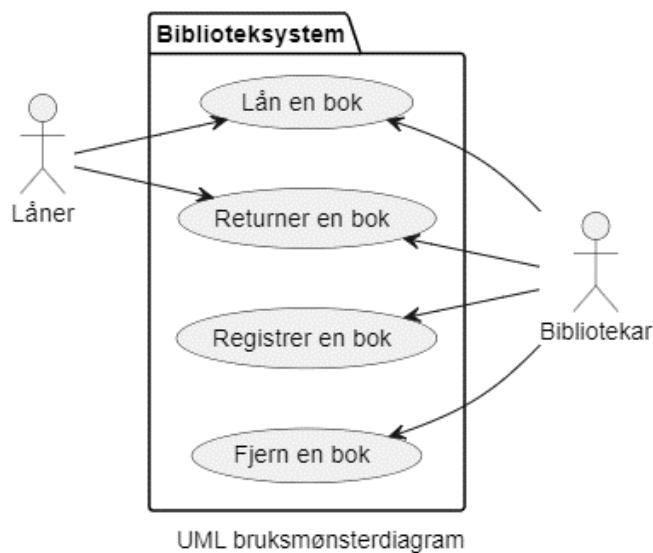
### for eksklusiv bruk ved <navn på skole>

I de neste avsnittene utvikler vi et forenklet bibliotekssystem for å vise hvordan bruksmønstre, klasse- og sekvensdiagrammer kan hjelpe oss til å strukturere programmet. Her er et eksempel på et bruksmønster for utlån av bok:

<b>Tittel:</b>	Lån en bok
<b>Aktører:</b>	Låner, Bibliotekar
<b>Hovedflyt:</b>	<ol style="list-style-type: none"><li>3. Låner oppgir tittel for å låne en bok</li><li>4. Bibliotekssystemet søker etter boken</li><li>5. Hvis boken er tilgjengelig:<ol style="list-style-type: none"><li>1. Boken registreres som utlånt</li><li>2. Bibliotekssystemet ber Bibliotekaren hente og gi boka til låner.</li></ol></li></ol>
<b>Alternativ flyt:</b>	<ol style="list-style-type: none"><li>3. Hvis boken ikke er tilgjengelig:<ol style="list-style-type: none"><li>1. Bibliotekssystemet informerer Låner om at boka er utilgjengelig</li></ol></li></ol>

### Bruksmønsterdiagram

UML bruksmønsterdiagram (*Use Case Diagram*) gir en oversikt over hvilke aktører som samhandler med systemets bruksmønstre. Disse diagrammene kan enkelt tegnes for hånd, eller vi kan også bruke verktøy som *plantUML* og *draw.io* til å lage dem. I *plantUML* kodes aktørene med `actor` og bruksmønstre med `usecase`. Bruksmønstrene kan legges inn i en `package` for systemet de tilhører.



```
@startuml Bibliotekssystem
left to right direction

actor "Låner" as L
actor "Bibliotekar" as B

package "Bibliotekssystem" {
    usecase "Lån en bok" as UC1
    usecase "Returner en bok" as UC2
    usecase "Registrer en bok" as UC3
    usecase "Fjern en bok" as UC4
}

L --> UC1
L --> UC2
UC1 <--> B
UC2 <--> B
UC3 <--> B
UC4 <--> B

@enduml
```

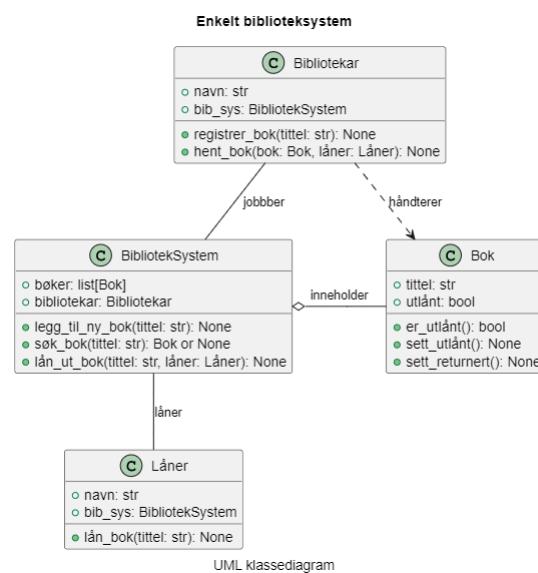
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Klassediagram

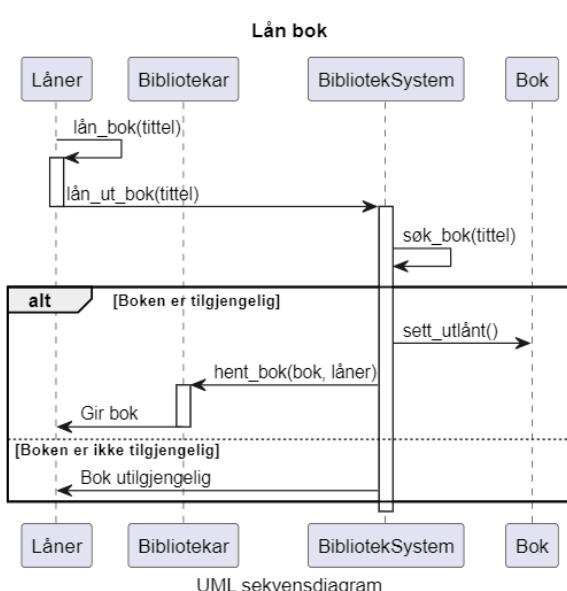
Ut fra det vi tidligere har lært om klassediagram og substantivmetoden, kan vi nå se på bruksmønstrene for å identifisere nøkkelklassene, som **Låner** og **Bibliotekar**. Disse klassene kommuniserer med den grunnleggende komponenten **BibliotekSystem**. Vi har også valgt å skille ut **Bok** som en egen klasse adskilt fra **BibliotekSystem**-klassen på grunn av enkeltansvarprinsippet som er en del av SOLID-prinsippene. Dette prinsippet understreker at en klasse bør ha ett og kun ett ansvar, noe som bidrar til at systemer blir mer modulære, enklere å vedlikeholde og lettere å forstå.

Bruksmønstrene gir oss også informasjon om hvilke metoder som trengs i de ulike klassene, og for å forstå hvordan klassene forholder seg til hverandre. Attributtene antar vi er selvforklarende.



### Sekvensdiagram

UML sekvensdiagram (*Sequence Diagram*) er en annen type UML-diagram vi kan bruke når vi skal planlegge et system. Mens bruksmønstre beskriver samspillet mellom brukere og systemet, viser sekvensdiagrammer hvordan deltagere (fortrinnvis objekter) kommuniserer med hverandre over tid, så vi kan se rekkefølgen av meldinger eller hendelser som sendes mellom objektene. Tiden går vertikalt fra toppen av diagrammet til bunnen og angis med en stiplet livslinje. Når et objekt blir aktivert, vises en avlang firkantet boks oppå livslinjen.



Aktørene **Låner** og **Bibliotekar** fra bruksmønsteret videreføres som deltagere i

```

@startuml biblioteksekvens
!pragma layout smetana
title Lån bok
caption UML sekvensdiagram

participant Låner
participant Bibliotekar
participant BibliotekSystem as bib_sys
participant Bok

Låner -> Låner : lån_bok(tittel)
activate Låner
Låner -> bib_sys : lån_ut_bok(tittel)
deactivate Låner
activate bib_sys
bib_sys -> bib_sys : søker_bok(tittel)
alt Boken er tilgjengelig
    bib_sys -> Bok : sett_utlånt()
    bib_sys -> Bibliotekar : hent_bok(bok, låner)
    activate Bibliotekar
    Bibliotekar -> Låner : Gir bok
    deactivate Bibliotekar
else Boken er ikke tilgjengelig
    bib_sys -> Låner : Bok utilgjengelig
end
deactivate bib_sys
@enduml

```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

sekvensdiagrammet, som nå også inkluderer **BibliotekSystem** og **Bok** for å modellere hele kommunikasjonsflyten.

Sekvensen **Lån bok** starter ved at låneren kaller sin egen `lån_bok(tittel)`. Å sende en melding vil i praksis ofte si å kalle en metode hos mottakeren. Når `lån_bok()` sender meldingen `lån_ut_bok(tittel)` til biblioteksystemet, er det en beskjed om å utføre denne metoden. Biblioteksystemet vil deretter sjekke tilgjengeligheten av boken ved å kalle sin egen `søk_bok()`. Hvis boken er tilgjengelig, vil bokens `sett_utlånt()` kjøres (slik at boka setter status på utlånt til `true`). Deretter får bibliotekaren en melding om å hente boken. Hvis boken er utilgjengelig, får låneren et svar fra biblioteksystemet om dette.

For å lage et sekvensdiagram i *PlantUML* bruker vi **participant** for å definere deltakere. Meldinger mellom deltakere representeres med piler (`->` for enveis melding og `-->` for en respons). **activate** og **deactivate** brukes for å vise aktivering og deaktivering av deltakere, og **alt** brukes for å vise alternativ flyt, avhengig av en betingelse..

### Fra OOM til OOP

Objektorientert modellering er abstrakt og på et høyere nivå enn objektorientert programmering. Modellene viser oss hvordan vi kan omdanne bruksmønstre, klasse- og sekvensdiagrammer til konkrete klasser og metoder. Vi vet dermed hvordan strukturen i programmet skal være før vi begynner å kode.

I eksempekkoden oppretter vi et biblioteksystem, en bibliotekar, to låner og tre bøker.

Biblioteksystemet har et **bibliotekar**- og et **bøker**-attributt. **Bibliotekar** og **Låner** har et **navn**- og **bib\_sys**-attributt, sistnevnte for å kunne kalle metodene i biblioteksystemet. Denne referansen bruker vi allerede i bibliotekarens `__init__()` for at biblioteksystemet skal få en referanse til bibliotekaren, som ikke fantes da biblioteksystemet ble opprettet. **Bok** har et **tittel**- og **utlånt**-attributt. Lånerens `lån_bok()` sender med tittelen og en referanse til seg selv når den kaller biblioteksystemet sin `lån_ut_bok()`, slik at vi vet hvem som skal låne boken. Slik fortsetter vi å følge sekvensdiagrammet. `lån_ut_bok()` kaller først `søk_bok()` i samme klasse. Hvis boken finnes og ikke er utlånt, kalles bokens `sett_utlånt()` og deretter bibliotekarens `hent_bok()`. Hvis boken ikke finnes i listen eller er utlånt, gis en beskjed om at den er utilgjengelig.

```
if __name__ == "__main__":
    # Opprett et bibliotek, en bibliotekar, og to låner
    bib_sys = BibliotekSystem()
    mathias = Bibliotekar("Mathias", bib_sys)
    emma = Låner("Emma", bib_sys)
    jonas = Låner("Jonas", bib_sys)

    # Register tre nye bøker i biblioteket
    mathias.registrer_bok("Python for nybegynnere")
    mathias.registrer_bok("Python for viderekomme")
    mathias.registrer_bok("Smidig IT-2")

    # Emma låner en bok
    emma.lån_bok("Python for viderekomme")

    # Jonas prøver å låne boken som Emma har lånt
    jonas.lån_bok("Python for viderekomme")
```

```
class Låner:
    def __init__(self, navn: str, bib_sys) -> None:
        self.navn = navn
        self.bib_sys = bib_sys

    def låن_bok(self, tittel: str) -> None:
        print(f'Låner: {self.navn} vil låne boken "{tittel}"')
        self.bib_sys.lån_bok(tittel, self)
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Vurderingsseksempler

```
class BibliotekSystem:
    def __init__(self) -> None: ...

    def legg_til_ny_bok(self, tittel: str) -> None: ...

    def søker_bok(self, tittel: str) -> Bok or None: ...

    def lån_bok(self, tittel: str, låner: Låner) -> None:
        bok = self.søker_bok(tittel) ←
        if bok and not bok.er_utlånt():
            bok.sett_utlånt() ←
            self.bibliotekar.hent_bok(bok, låner) ←
            print(
                f'BibliotekSystem: Boken "{tittel}" er lånt ut til {låner.navn}.')
        else:
            print(
                f'BibliotekSystem: Boken "{tittel}" er ikke tilgjengelig for {låner.navn}.')


```

```
class BibliotekSystem:
    def __init__(self) -> None: ...

    def legg_til_ny_bok(self, tittel: str) -> None: ...

    def søker_bok(self, tittel: str) -> Bok or None:
        print(f'BibliotekSystem: Søker etter boken "{tittel}".')
        for bok in self.bøker:
            if bok.tittel == tittel:
                return bok
        return None

    def lån_bok(self, tittel: str, låner: Låner) -> None: ...


```

```
class Bok:
    def __init__(self, tittel: str) -> None: ...

    def er_utlånt(self) -> bool: ...

    def sett_utlånt(self) -> None:
        self.utlånt = True
        print(f'Bok: "{self.tittel}" er registrert som utlånt.')

    def sett_returnert(self) -> None: ...


```

```
class Bibliotekar:
    def __init__(self, navn: str, bib_sys: BibliotekSystem) -> None: ...

    def registrer_bok(self, tittel: str) -> None: ...

    def hent_bok(self, bok: Bok, låner: Låner) -> None:
        print(
            f'Bibliotekar: {self.navn} gir boken "{bok.tittel}" til {låner.navn}.')


```

```
Bibliotekar: Mathias legger boken "Python for nybegynnere" inn i systemet.
BibliotekSystem: Boken "Python for nybegynnere" er registrert.
Bibliotekar: Mathias legger boken "Python for viderekomne" inn i systemet.
BibliotekSystem: Boken "Python for viderekomne" er registrert.
Bibliotekar: Mathias legger boken "Smidig IT-2" inn i systemet.
BibliotekSystem: Boken "Smidig IT-2" er registrert.
Låner: Emma vil låne boken "Python for viderekomne".
BibliotekSystem: Søker etter boken "Python for viderekomne".
Bok: "Python for viderekomne" er registrert som utlånt.
Bibliotekar: Mathias gir boken "Python for viderekomne" til Emma.
BibliotekSystem: Boken "Python for viderekomne" er lånt ut til Emma.
Låner: Jonas vil låne boken "Python for viderekomne".
BibliotekSystem: Søker etter boken "Python for viderekomne".
BibliotekSystem: Boken "Python for viderekomne" er ikke tilgjengelig for Jonas.
```

Det er lett å se for seg mulige utvidelser for dette enkle biblioteksystemet. Vi kunne lagt til brukerregistrering og innlogging, utfyllende bokinformasjon som forfatter, ISBN-nummer og anmeldelser, digitale kvitteringer for returnerte bøker, samt en historikk over utlån og ikke minst feilhåndtering. Videre kunne det være nyttig å vite om en utilgjengelig bok er utlånt

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

eller ikke finnes i systemet, ventelister, forfallsdatoer, purringer, kommunikasjon med andre biblioteker, osv.

Bruksmønstre og sekvensdiagrammer er viktige verktøy for å analysere og utforme systemets klasser og funksjonalitet, bedre enn om vi bare hadde sett på substantiver og verb i en kravspesifikasjon.

#### **Ta med minst dette**

Når vi trenger et datasystem, bør vi vurdere om vi skal utvikle det selv eller kjøpe et ferdig system. Før utviklingen må vi lage en kravspesifikasjon som dokumenterer alle krav til systemet. I praksis omfatter dette mye mer enn bare funksjonelle krav. Bruksmønstre beskriver hvordan brukere samhandler med systemet. Bruksmønsterdiagrammer gir en oversikt over hvilke aktører som samhandler med systemets bruksmønstre. Bruksmønstre er i seg selv tekstdokumenter. Klassediagrammer viser systemets statiske struktur, mens sekvensdiagrammer viser hvordan objekter samhandler dynamisk. Kravspesifikasjon og bruksmønstre hører typisk til analysefasen for å beskrive hva som skal gjøres og hvordan systemet skal brukes. Klassediagrammer og sekvensdiagrammer er derimot en del av designfasen for å beskrive hvordan systemet skal kodes.

#### **Øvingsoppgaver**

##### **4.7.1**

Bruksmønstre og pseudokode har flere fellestrek. Begge teknikkene

- bruker et språk som er lett å forstå
- bidrar som brobygger mellom de som koder og de som ikke gjør det
- gir en trinnvis beskrivelse av systemets funksjonalitet
- foregår før den faktiske koding begynner

Men hva skiller dem fra hverandre?

##### **4.7.2**

Ta utgangspunkt i det enkle biblioteksystemet

- a) Lag et bruksmønster for tilfellet "Returner en bok"
- b) Oppdater klassediagrammet for det enkle biblioteksystemet
- c) Lag et sekvensdiagram for tilfellet "Returner en bok"
- d) Implementer koden for "Returner en bok"

## 4.8 Mer om testing

### Bolkens innhold

Et tilbakeblikk.....	300
Organisering og gjenbruk av kode.....	301
Integrasjonstesting .....	303
GUI-testing.....	303
Flere emner innen testing .....	306
Ta med minst dette.....	306
Øvingsoppgaver.....	306

### Kompetansemål:

- vurdere og bruke strategier for feilsøking og testing av programkode

### Et tilbakeblikk

Et av formålene med programvaretesting er å identifisere og rette feil. I bolken [1.7 Feil](#) utforsket vi ulike feiltyper i programmering, som syntaksfeil, semantiske feil og logiske feil. Vi lærte at syntaksfeil blir fanget opp av *Pylance* i Visual Studio Code og markert med røde bølgete linjer. Dermed kan vi rette disse feilene før programmet kjøres. Semantiske og logiske feil blir vanligvis først avslørt under kjøring av programmet. Disse feilene kan vi oppdage med tester, før programmet tas i bruk, som ved å bruke `assert`-setninger. Når vi oppdager en feil, har vi flere hjelpemidler til rådighet for å rette den. Ofte gir feilmeldingene oss tilstrekkelig informasjon, eller vi klarer oss med noen midlertidige `print`-setninger for å kontrollere verdiene til noen utvalgte variabler. Andre ganger må vi gå grundigere til verks og bruke et feilsøkingsprogram (*debuggeren*). Noen feil er uunngåelige, som når vi mottar feilaktige data fra brukere eller andre systemer. Dette kan vi ta høyde for med unntakshåndtering (`try-except`), som sikrer at programmet kan fortsette å kjøre selv når slike situasjoner oppstår.

I bolken [2.5 Programvaretesting](#) kom det tydelig fram at testing spiller en avgjørende og omfattende rolle i systemutviklingsprosessen. Vi diskuterte tester på ulike nivåer, som enhetstesting, integrasjonstesting, systemtesting og akseptansestesting. Vi så også hvor viktig det er å ha en strukturert testplan, og vi lærte å bruke *Pytest* for å skrive våre første enhetstester for å sjekke at koden fungerer som forventet. I tillegg introduserte vi testdrevet utvikling (TDD), en metode som innebærer å skrive tester før selve koden for å fremme bedre kodekvalitet og pålitelighet. Vi så også hvordan vi kunne effektivisere testene med parametere, og hvordan vi kunne markere tester for å kategorisere dem og kontrollere hvilke tester som skal kjøres.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

I bolken [3.7 Enhetstesting av objektorienterte programmer](#) utforsket vi flere detaljer om testing av objektorienterte programmer. Vi lærte at et testtilfelle i `pytest` er en funksjon som starter med `test_`, og en testsamling er en `.py`-fil som også starter med `test_` og inneholder flere slike testfunksjoner. Vi så spesielt på testing av klasser og metoder, inkludert opprettelse og initialisering av objekter, metoder som returnerer verdier, metoder med bivirkninger og metoder som håndterer unntak. Vi brukte `fixtures` i `pytest` for å sette opp og rydde opp testkonteksten. I andre testverktøy, som `unittest`, gjøres dette med `setUp-` og `tearDown-` metoder. Dette er vist som tilleggsstoff i [5.7 Unittest](#). Vi refaktorerte koden etter testing for å gjøre den enklere og mer lesbar, og vi dokumenterte testene. Etterligning (*mocking*) vil si å erstatte deler av koden med imitasjoner når vi tester kode som er avhengig av uforutsigbare data, som løpende aksjekurser. Det blir brukt senere i denne bolken under [GUI-testing](#) og er også vist som tilleggsstoff i [5.6 Etterligning \(mocking\) med pytest](#). Vi brukte `pytest` fordi det er enkelt, brukervennlig og populært.

#### Organisering og gjenbruk av kode

Formålet med denne bolken er å lære mer om integrasjonstesting og GUI-testing. Når det gjelder GUI-testing, er det viktig å skille brukergrensesnittet fra resten av applikasjonen. Det gjør det lettere å teste både brukergrensesnittet og programmets logikk hver for seg. Integrasjonstesting handler om samspillet mellom ulike komponenter. Derfor må vi forstå hvordan koden kan organiseres, de ulike komponentenes roller, funksjoner og hvordan de samarbeider.

**Funksjoner** er de mest grunnleggende enhetene for organisering av kode. De lar oss kapsle inn kode for en bestemt oppgave. Dermed kan denne koden gjenbrukes og kalles flere ganger gjennom programmet. Funksjoner bidrar samtidig til at koden blir enklere og mer lesbar.

**Klasser** er uunnværlige i objektorientert programmering (OOP) og lar oss opprette objekter med egenskaper (attributter) og handlinger (metoder) definert i klassen. De hjelper oss med å modellere komplekse systemer på en forståelig måte, ofte ved å representere virkelige objekter.

**Metoder** er funksjoner inne i en klasse. De kan bruke og endre objektets egne data, og bestemmer hva objektet kan gjøre.

**Moduler** er `.py`-filer som inneholder Python-kode, ofte i form av funksjoner og klasser med tilhørende attributter og metoder. Elementene i en modul er tilgjengelige for hverandre, og når en modul importeres i en annen, blir disse elementene også tilgjengelig i den nye modulen. Dette fremmer mer gjenbruk av kode og bidrar til en mer modulær oppbygging med de fordeler det medfører.

For å importere en modul, må den enten være installert i Python-miljøet eller være tilgjengelig i prosjektets filstruktur. Når vi importerer en modul, bruker vi navnet på `.py`-filen uten `.py`-endelsen. Moduler som er installert globalt med `pip` kan adresseres (absolutt) med bare modulens navn, for banen håndteres automatisk av Python. Moduler som tilhører vårt

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

eget prosjekt må importeres med relativ adressering, med bruk av punktum (.) for å indikere filens plassering i forhold til gjeldende modul.

**Pakker** i Python er mapper som inneholder moduler og en spesiell fil kalt `__init__.py`. Denne filen kan være tom, men den kan også inneholde initialiseringskode, importsetninger og globale variabler eller konstanter. Å bruke pakker framfor vanlige mapper som bare inneholder moduler, gir flere fordeler, særlig i større prosjekter. Blant annet gir det bedre organisering og struktur, hjelper med å unngå navnekonflikter mellom moduler, og forenkler importprosessen ved å tillate import av hele pakker eller spesifikke moduler.

Navnet på pakken er navnet på mappen som inneholder modulene og `__init__.py`-filen. Når vi skal bruke pakken i et prosjekt, må vi først forsikre oss om at Python kan finne pakken. Dette kan vi gjøre ved å installere pakken globalt, legge mappen hvor pakken ligger til miljøvariabelen `PYTHONPATH`, eller bruke relativ import.

**Biblioteker** i Python er ikke like klart definert som de enhetene vi har beskrevet ovenfor. Dessuten brukes begrepene "pakke" og "bibliotek" ofte om hverandre. Mens pakker er gjenkjennelige på grunn av sin struktur (moduler og `__init__.py` i en mappe), forbindes biblioteker mer med den funksjonaliteten de tilbyr. Kode vi skriver selv, kan vi organisere i pakker. Kode vi bruker fra andre, henter vi ofte fra installerte biblioteker. Med litt forarbeid kan pakker publiseres, for eksempel til [PyPI](#), hvorfra de kan lastes ned, installeres og brukes som biblioteker av andre utviklere. Et bibliotek tilbyr oss byggesteiner (funksjoner, klasser og metoder organisert i moduler og pakker) som vi kan bruke i programmene våre, i motsetning til å installere ferdige Python applikasjoner. I Smidig IT-2 har vi brukt mange biblioteker, blant annet [requests](#), [tkinter](#), [pandas](#), [seaborn](#), og [pygame](#).

**Rammeverk** i Python tilbyr mer enn det vi finner i moduler, pakker og biblioteker. Selv om skillet mellom et bibliotek og et rammeverk kan være flytende, gir rammeverk en strukturert arkitektur for hele applikasjoner, komplett med innebygget funksjonalitet tilpasset denne strukturen. Dette gjør det enklere og raskere å utvikle applikasjoner for spesifikke formål, men kan også medføre en viss grad av begrensning. [Flask](#) og [Dash](#), som utforskes i [5.11 API for web og mobil med Flask og Dash](#) (Tilleggsstoff), er eksempler på rammeverk spesielt utviklet for å lage webapplikasjoner. [Tkinter](#) kan betraktes som et bibliotek når vi velger ut og benytter GUI-komponenter til programmene våre, men det kan også ses som et rammeverk, ettersom det krever en bestemt struktur med å opprette et vindu ([Tk](#)) og [mainloop\(\)](#) for å kjøre applikasjonen. På samme måte kan kombinasjonen av [Jupyter](#) og [Pandas](#) betraktes som et rammeverk for dataanalyse, der interaktiv utforskning innebærer å bruke Pandas for effektiv datahåndtering, samtidig som integrert dokumentasjon og visualisering inngår i den samlede metodikken. Selv om Pygame i seg selv er et bibliotek, begynner det å ligne en form for rammeverk når vi tar i bruk [den objektorienterte malen for Pygame](#), som vi introduserte i bolken [3.4 Tekst, grafikk, animasjon og lyd med Pygame](#). Denne malen gir en ferdigprogrammert struktur for håndtering av hendelser, oppdateringer og grafikk, noe som gjør det raskt og enkelt å komme i gang med å utvikle spill.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Integrasjonstesting

Integrasjonstesting innebærer å kombinere individuelle moduler eller komponenter i et system for å sikre at de fungerer sammen som forventet. I motsetning til enhetstesting, som tester enkelte komponenter isolert, har integrasjontesting som mål å avdekke feil i samspillet mellom ulike deler av et program. Dette er spesielt viktig i utviklingen av komplekse applikasjoner.

I vårt enkle eksempel har vi en **Spiller**-klassen definert i én modul og en **PoengSystem**-klassen i en annen. **Spiller**-klassen representerer en spiller som kan utføre ulike handlinger, for eksempel "levert\_sau", som gir poeng. **PoengSystem**-klassen er ansvarlig for å holde styr på og oppdatere spillerens poengsum basert på handlingene utført av spilleren. Integrasjonstester undersøker hvordan disse to klassene samspiller. Her tester vi spesifikt om poengene oppdaterer korrekt når spilleren utfører handlingene "levert\_sau" og "ingen\_handling". Se mappen **b\_4\_8\_1\_integrasjonstesting**.

```
# spiller.py
from poengsystem import PoengSystem

class Spiller:
    def __init__(self, navn):
        self.navn = navn
        self.poengsystem = PoengSystem()

    def oppdater_poeng(self, handling):
        self.poengsystem.oppdater_poeng(handling)
```

```
# poengsystem.py
class PoengSystem:
    def __init__(self):
        self.poeng = 0

    def oppdater_poeng(self, handling):
        if handling == "levert_sau":
            self.poeng += 1

    def hent_poeng(self):
        return self.poeng
```

```
# test_integrasjon.py
import pytest
from spiller import Spiller

@pytest.fixture
def spiller_ola():
    return Spiller("Ola")

def test_poeng_for_levert_sau(spiller_ola):
    spiller_ola.oppdater_poeng("levert_sau")
    assert spiller_ola.poengsystem.hent_poeng() == 1

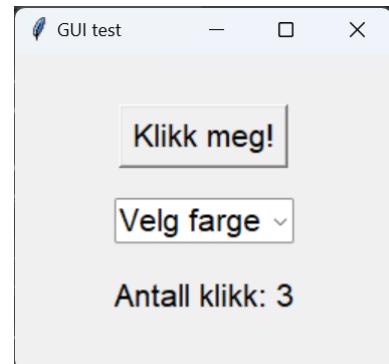
def test_poeng_for_ingen_handling(spiller_ola):
    spiller_ola.oppdater_poeng("ingen_handling")
    assert spiller_ola.poengsystem.hent_poeng() == 0
```

### GUI-testing

GUI-testing innebærer vurdering av brukergrensesnittets estetikk, funksjonalitet og brukervennlighet, noe som kan være vanskelig å automatisere fullstendig. Det vil være nødvendig med manuelle tester for å kvalitetssikre brukeropplevelsen og det visuelle designet, som fargekonsistens, skrifttyper, og hvor intuitivt brukergrensesnittet er, noe som krever subjektive vurderinger og menneskelig innsikt. Men vi kan automatisere testingen av knapper, menyer, dialogbokser, med mer. Når vi skal teste funksjonaliteten, er det lurt å skille brukergrensesnittet fra resten av programmet. Det gjør programmet også lettere å vedlikeholde og utvide samt at programlogikken kan gjenbrukes for ulike brukergrensesnitt, enten det skulle være et webgrensesnitt, en mobilapp eller et desktop-program. Denne praksisen med å skille brukergrensesnittet fra programlogikken er så utbredt at det har blitt utviklet flere designmønstre for å støtte dette, hvorav MVC (*Model-View-Controller*) er et av de mest kjente. Se også [5.8 Designmønstre](#) (tilleggsstoff).

Her har vi tatt utgangspunkt i [Tkinter App-GUI malen](#) som ble presentert i [3.9 Reelle datasett med OOP og GUI](#). Den gjør nettopp dette, men slår sammen *Model*- og *Controller*-klassen i en **App**-klasse. **Gui**-klassen tilsvarer *View*-klassen og kan testes uavhengig resten av programmet. Selve programmet har en knapp (`tk.Button`) og en combobox (`ttk.Combobox`) som kan oppdatere teksten i en etikett (`tk.Label`). Se filen `gui_testing.py` i mappen

`\eksempler\4_O8_fortsettelse_av_testing\b_4_8_2_gui_testing`.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Vi kan nå utføre testing av `Gui`-klassen isolert ved hjelp av `pytest`. For å gjøre dette, må vi etterlignige `App`-klassen med en *mock* siden initialiseringen av `Gui`-klassen krever en `App`-instans. Ved å bruke `mocker-fixture` fra `pytest-mock`, sikrer vi at `Gui`-klassen kan initialiseres og testes uten å involvere logikken i `App`-klassen. `MagicMock` sørger for at metodene ikke kjøres, men vi kan kontrollere både at de blir kalt og hvilke argumenter som ble brukt. `mocker-fixture` blir tilgjengelig i `pytest` når vi installerer `pytest-mock` (`pip install pytest-mock`). Se filen `test_gui.py`.

Ved å bruke `assert <komponent>.winfo_exists()` sikrer vi at komponentene både er initialisert og eksisterer i brukergrensesnittet, noe som er bedre enn bare `assert <komponent> is not None`. Deretter tester vi om komponentene er riktige typer, og hvis vi ønsker, kan vi også forsikre oss om at de har riktig posisjon, farge, osv. ved å bruke metoder som `winfo_x()`, `winfo_y()` og `cget('background')`.

`update`-metoden er enkel å teste. Vi gir den en verdi for enten teller eller farge, og deretter sjekker vi at teksten i etiketten er korrekt oppdatert.

Siden `app` er en `MagicMock`, kan vi kontrollere at knappen kaller `on_click()` ved å simulere et klikk med `invoke()` og forsikre oss om at metoden ble kalt med `assert_called_once()`. `invoke()` er en metode i Tkinter som brukes til å simulere et klikk på en knapp, uavhengig av om det er i et testprogram eller ikke, mens `assert_called_once` er en metode i `MagicMock`-klassen spesielt laget for testing.

Når vi tester metoden `endret_farge`, oppdaterer vi `farge_valg`, som er en `StringVar`, ved å sette verdien til 'Grønn'. Når `farge_valg` endres, utløses `endret_farge()` automatisk på grunn av `trace_add`-bindingen. Vi forventer at `endret_farge()` kaller `on_select()` i `App`-klassen og får dette bekreftet gjennom `assert_called_once_with("Grønn")`. Selv om vi ikke kan teste direkte at `endret_farge()` blir kalt fordi den ikke er en *mock*, har vi kontrollert at koden utføres som forventet og har dermed en indirekte bekreftelse på at `trace_add`-bindingen fungerer korrekt.

```
@pytest.fixture()
def gui(mocker):
    app = mocker.MagicMock()
    gui = Gui(app)
    yield gui
    gui.destroy()
```

```
def test_create_widgets(self, gui):
    # Test at widgetene er opprettet som forventet
    assert gui.winfo_exists()
    assert gui.frame.winfo_exists()
    assert isinstance(gui.frame, tk.Frame)
    assert gui.button.winfo_exists()
    assert isinstance(gui.button, tk.Button)
    assert gui.combo.winfo_exists()
    assert isinstance(gui.combo, ttk.Combobox)
    assert gui.combo.cget("values") == ("Rød", "Grønn", "Blå")
    assert gui.combo.get() == "Velg farge"
    assert gui.label.winfo_exists()
    assert isinstance(gui.label, tk.Label)
    assert gui.label.cget("text") == "Antall trykk: 0"
```

```
def test_update(self, gui):
    # Test at update-metoden oppdaterer labelen som forventet
    gui.update(verdi=42, type="teller")
    assert gui.label["text"] == "Antall trykk: 42"
    gui.update(verdi="Grønn", type="farge")
    assert gui.label["text"] == "Valgt farge: Grønn"
```

```
def test_button(self, gui):
    # Test at knappen oppfører seg som forventet
    gui.button.invoke()
    gui.app.on_click.assert_called_once()
```

```
def test_endret_farge(self, gui):
    # Test at endret_farge-metoden oppfører seg som forventet
    gui.farge_valg.set("Grønn")
    gui.app.on_select.assert_called_once_with("Grønn")
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Når vi tester *comboboxens* oppførsel, sjekker vi at et valg oppdaterer `farge_valg`, og at denne oppdateringen også utløser `endret_farge()`. Dette bekrefter at `trace_add`-bindingen mellom `farge_valg` og `endret_farge()` fungerer som forventet, noe vi indirekte fikk bekreftet i forrige test.

```
def test_combobox(self, mocker):
    # Test at komboboksen oppfører seg som forventet.
    # Mock endret_farge før denne bindes til farge_valg
    # med trace_add når Gui-instansen opprettes.
    mock_endret_farge = mocker.patch.object(
        Gui, "endret_farge", autospec=True)
    app = mocker.MagicMock()
    gui = Gui(app)
    gui.combo.set("Grønn")
    assert gui.farge_valg.get() == "Grønn"
    mock_endret_farge.assert_called_once()
    args = mock_endret_farge.call_args[0]
    assert "write" in args
```

Vi må etterligne `endret_farge()` før den bindes til `farge_valg` med `trace_add()` under opprettelsen av `Gui`-instansen. Hvis vi prøver å gjøre dette etterpå, vil den virkelige `endret_farge()` bli kalt i stedet, så dette er grunnen til at vi ikke bruker `gui`-fixturen i denne testen. Her har vi valgt å bruke `mocker.patch.object` i stedet for `MagicMock` fordi vi får samme signatur som den opprinnelige metoden med `autospec=True`.

Merk at argumentene (`*args`) til `endret_farge()` er tre tekststrenger, hvor den siste er `'write'`. Når metoden er en *mock*, får vi tak i verdiene til disse argumentene gjennom `call_args[0]`, som er en *tuple*. Derfor bruker vi `assert "write" in args`. Selve fargen er ikke med i `*args` og må hentes med `farge_valg.get()` i den virkelige `endret_farge()`, noe vi verifiserte i `test_endret_farge()`.

Vi er nå ferdige med testingen av brukergrensesnittet. Enhetstest av `App`-klassen ligger i `test.app.py` og vil ikke bli gjennomgått her da dette skal være kjent fra før. Vi kan imidlertid kontrollere integrasjonen mellom `App`-klassen og `Gui`-klassen. `app-fixturen` oppretter en instans av `App`-klassen, som automatisk også oppretter en instans av `Gui`-klassen. Se filen `test_app_gui.py`.

```
@pytest.fixture()
# Opprett en instans av App (med Gui) for testing.
def app():
    app = App()
    yield app
    app.gui.destroy()
```

Knappen er konfigurert med en `command`-parameter som angir *callback*-funksjonen `on_click` i `app`-instansen. Det kalles en tilbakekallingsfunksjon fordi den blir kalt senere når en spesiell hendelse inntreffer. Vi kan simulere denne hendelsen med `invoke`, som etterligner et klikk på knappen. Deretter kontrollerer vi at både `teller`-attributtet i `app`-instansen og `text`-attributtet i `label`-instansen i `gui`-instansen er oppdatert korrekt.

```
def test_button_click(self, app):
    # Test at teller oppdatters i app
    # og label i gui
    app.gui.button.invoke()
    assert app.teller == 1
    assert app.gui.label.cget("text") == "Antall klikk: 1"
```

*Comboboxen* er konfigurert med en `textvariable`-parameter som binder `farge_valg`, en `StringVar`, til `combo`-objektet. I `test_combobox_select` setter vi fargen til "Blå" med `app.gui.combo.set("Blå")`. Dette oppdaterer `farge_valg` og utløser `endret_farge()` på grunn av `trace_add`-bindingen. Metoden `endret_farge` kaller deretter `on_select()` i `app`-instansen. `on_select` oppdaterer først sitt `valgt_farge`-attributt og deretter kaller den `update`-metoden i `gui`-instansen. Dette oppdaterer teksten til etiketten. Så etter å ha valgt fargen i *comboboxen*, trenger vi bare å forsikre oss om at både `valgt_farge`-attributtet i `app`-instansen og `text`-attributtet til `label`-instansen i `gui`-instansen har blitt oppdatert som forventet. Resten går av seg selv.

```
def test_combobox_select(self, app):
    # Test at valgt_farg oppdateres i app
    # og label i gui
    app.gui.combo.set("Blå")
    assert app.valgt_farge == "Blå"
    assert app.gui.label.cget("text") == "Valgt farge: Blå"
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Fleire emner innen testing

Testing er et eget fagfelt innen programvareutvikling, og det er fullt mulig å bygge en langsigkt karriere innen testing. Det vil normalt kreve en utdanning innen informatikk og/eller kvalitetssikring samt ulike sertifiseringer. Vi har lært å lage tester på ulike nivåer, som enhetstesting, nå også med GUI-testing av brukergrensesnittet, og integrasjonstesting. Tidligere har vi også omtalt systemtesting og akseptansestesting, men fagfeltet er betydelig bredere. Det omhandler også testplanlegging, ytelsestesting, sikkerhetstesting, testautomatisering, kontinuerlig integrasjon og kontinuerlig levering (CI/CD), målinger og rapportering, testing i ulike systemutviklingsmetoder, etisk hacking og penetrasjonstesting, samt en rekke andre områder som testdatabehandling, bruk av testverktøy og rammeverk, opprettelse av testmiljøer, testdesignmønstre og utforskende testing. Hvert av disse områdene omfatter igjen mange temaer. Eksempelvis når vi ser på testrapporter, handler det om mye mer enn kun å dokumentere testresultatene. Det kan rapporteres om testdekning, feilfrekvens, testtid, testeffektivitet, feilrettetid, kostnad per feil, kodekvalitet i form av kompleksitet, unødvendig duplisering og lesbarhet, osv. Så selv om utviklere lager tester for sine egne applikasjoner, er testing definitivt et eget fagfelt med et bredt spekter av spesialiseringer og problemstillinger.

### Ta med minst dette

Vi begynte med et tilbakeblikk på hva vi hadde lært om testing før denne bolken. Det var hovedsakelig å forstå ulike typer feil, teori om programvaretesting og praksis i enhetstesting av objektorienterte programmer. I denne bolken har vi utforsket integrasjonstesting og GUI-testing, og da er det viktig å forstå hvordan vi kan organisere koden for effektiv testing som også fremmer gjenbruk og gjør den lettere å lese. Integrasjonstesting sikrer at ulike enheter fungerer sammen som forventet. I GUI-testing skiller vi brukergrensesnittet fra programmets logikk for å kunne teste delene hver for seg. Automatisering av GUI-tester kan effektivisere prosessen, selv om noen manuelle tester fortsatt er nødvendige for å sikre en god brukeropplevelse. Testing er et eget fagfelt med mange spesialiseringsmuligheter, og det dekker langt flere aktiviteter enn det vi har hatt anledning til å utforske.

### Øvingsoppgaver

#### 4.8.1

I mappen `\eksempler\4_08_fortsettelse_av_testing\p_4_8_1_oppgave` ligger en enkel applikasjon som utfører og logger banktransaksjoner. `konto.py` inneholder en `Konto`-klasse med metoder for `innskudd` og `uttak`, og `logg.py` inneholder en `Logg`-klasse for logging av transaksjoner. Enhetstester for `Konto`- og `Logg`-klassen er allerede skrevet (`test_konto.py` og `test_logg.py`).

- Skriv og kjør en integrasjonstest mellom `Konto`- og `Logg`-klassen.

### Test Manager

Arbeidsgiver: Kongsberg Maritime - Automation & Control



KONGSBERG

Stillingstittel: Test Manager

Frist: 14.01.2024

Ansettelsesform: Fast

Are you Handelsbanken's digital unit's new Sr. Test Automation Engineer?

Handelsbanken

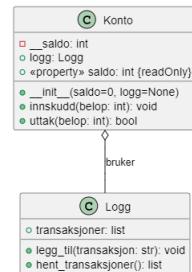
Arbeidsgiver: Handelsbanken Norge

Stillingstittel: Sr. Test Automation Engineer

Frist: 14.01.2024

Ansettelsesform: Fast

Vurderingsseksemplar



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

4.8.2



I mappen  
\\eksempler\\4\_08\_fortsettelse\_av\_testing\\p\_4\_8\_2\_oppgave  
ligger en enkel applikasjon som viser temperaturene i  
tre norske byer. `temperatur.py` inneholder en ordbok  
med temperaturene for Oslo, Bergen og Trondheim.

Brukeren kan velge en by ved hjelp av radioknapper i brukergrensesnittet, og applikasjonen viser temperaturen for den valgte byen. Det er allerede skrevet en enhetstest for `App`-klassen, `test_app.py`.

- Skriv og kjør en enhetstest for `Gui`-klassen
- Skriv og kjør en test for integrasjonen mellom `App`- og `Gui`-klassen.

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 4.9 Reelle datasett med REST API

#### Bolkens innhold

Befolkingstall (SSB).....	308
Prosedyreorienterte og objektorienterte programmer.....	310
REST-API.....	311
Valutakurser (NB).....	312
Ta med minst dette.....	314
Øvingsoppgaver.....	315

#### Kompetanse mål:

- bruke programering til å innhente, analysere og presentere informasjon fra reelle datasett
- utvikle objektorienterte programmer med klasser, objekter, metoder og arv
- generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode
- utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger

#### Befolkingstall (SSB)

I [bolk 1.9](#) brukte vi reelle datasett med CSV-format, og i [bolk 2.9](#) brukte vi JSON-format. Vi lastet ned hele datasettet. Nå skal vi bruke REST API og hente deler av datasettet, tilsvarende spørrsager i en database. I tabell [01222](#) har Statistisk sentralbyrå informasjon om befolkningen i hver kommune.

Her har vi plukket ut [Befolking ved utgangen av kvartalet, 2024K1](#) og kommune [K-3101 Halden](#).

Vi finner at befolkningen i Halden var 31 965 den gangen. API-spørringen for denne tabellen får vi også. La oss se om vi kan hente

#### Befolking

The screenshot shows the API interface for selecting variables, time periods, and regions for the population dataset. The selected parameters are:

- Statistikkkvariabel: K-3101 Halden
- Kvartal: 2024K1
- Region: Kommuner 2020, sammenlæste tidsperiode

The result table shows the population for Halden in 2024K1, which is 31 753. Below the table, the JSON query used for the API call is displayed:

```
Send (POST) følgende JSON-spørring til URL under eller kopier til API-konsoll.  
URL:  
https://data.ssb.no/api/v0/no/table/01222/  
JSON-spørring:  
[{"query": {  
    "code": "Region",  
    "selection": {},  
    "filter": "agg:KommSummer",  
    "values": [  
        "K-3101"  
    ]  
},  
{  
    "code": "ContentsCode",  
    "selection": {  
        "filter": "item",  
        "values": [  
            "Folketalletell"  
        ]  
    }  
}]
```

#### Befolking

The screenshot shows the API interface for selecting variables, time periods, and regions for the population dataset. The selected parameters are:

- Statistikkkvariabel: K-3101 Halden
- Kvartal: 2024K1
- Region: Kommuner 2020, sammenlæste tidsperiode

The result table shows the population for Halden in 2024K1, which is 31 753. Below the table, the JSON query used for the API call is displayed:

```
Send (POST) følgende JSON-spørring til URL under eller kopier til API-konsoll.  
URL:  
https://data.ssb.no/api/v0/no/table/01222/  
JSON-spørring:  
[{"query": {  
    "code": "Region",  
    "selection": {},  
    "filter": "agg:KommSummer",  
    "values": [  
        "K-3101"  
    ]  
},  
{  
    "code": "ContentsCode",  
    "selection": {  
        "filter": "item",  
        "values": [  
            "Folketalletell"  
        ]  
    }  
}]
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

det samme fra et Python-program ved å bruke [veiledningen](#) til SSB.

Kopier programmet fra veiledningen og bytt ut URL-en og JSON-spørringen med opplysningene fra befolkningstabellen. Den siste linjen i programmet har vi erstattet med `print(df)` for å se at vi får ut [Halden](#) og [31965](#).

Se [b\\_4\\_09\\_0\\_1\\_fra\\_veiledning.py](#)

```
print(resultat)

dataset = pyjstat.Dataset.read(resultat.text)
df = dataset.write("dataframe")
print(df)
✓ 0.5s

<Response [200]>
  region                      statistikkvariabel kvartal value
 0 Halden  Befolking ved utgangen av kvartalet 2024K1 31965
```

Hvis vi bruker kommunenummeret [K-3103](#), ser vi at det var [52213](#) innbyggere i [Moss](#).

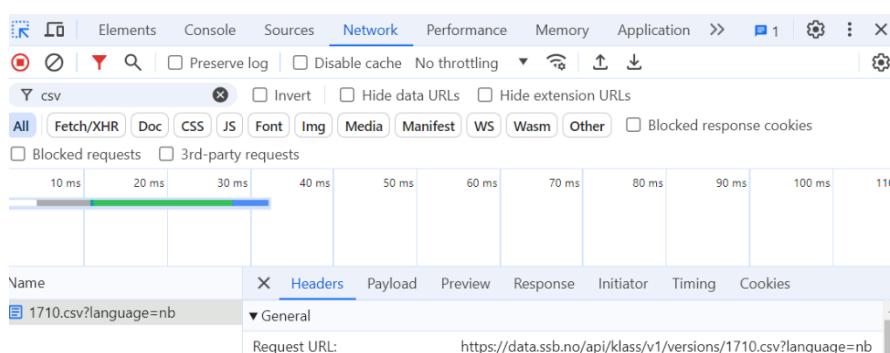
Hele programmet kan gjøres om til en funksjon `hent_befolknings` med `commune_nr` som parameter. Her kaller vi funksjonen med [K-3105](#) som argument og ser at det var [59906](#) innbyggere i [Sarpsborg](#). I stedet for å returnere en `DataFrame`, returnerer vi en ordbok (`dict`) med `commune` og `befolknings`.

Se [b\\_4\\_09\\_0\\_2\\_befolknings.py](#)

Hvis vi skal lage et program hvor vi velger kommune og får tilbake opplysninger om antall innbyggere, må vi finne en kobling mellom `commune` og `commune_nr`.

### Kommuneinndelingen

finnes også på SSBs websider. Vi kan laste ned CSV-filen, eller bruke F12 og Nettverkfanen i Chrome til å finne denne [lenken](#). Vi må bruke `encoding="ISO-8859-1"` når vi leser filen med bruker `read_csv`, ellers vil det oppstå feil.



```
import pandas as pd
url ="https://data.ssb.no/api/klass/v1/versions/1710.csv?language=nb"
df = pd.read_csv(url, sep=";", encoding="ISO-8859-1")
display(df.head(3))
```

code	parentCode	level	name	shortName	notes	validFrom	validTo
0	301	Nan	1	Oslo	Nan	I 1980 ble del av 0231 Skedsmo overført til 03...	Nan
1	1101	Nan	1	Eigersund	Nan	I 1967 ble 1101 Eigersund del overført til 111...	Nan
2	1103	Nan	1	Stavanger	Nan		Nan

Vi trenger bare kolonne [0](#) og [3](#), og lager en ny funksjon `hent_kommune_nr`, som returnerer disse kolonnene med nye navn, `commune_nr` og `commune`. Se [b\\_4\\_09\\_0\\_3\\_kommune.py](#)

```
df = df[['code', 'name']]
df.columns = ['commune_nr', 'commune']
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Prosedyreorienterte og objektorienterte programmer



Nå er alt klart for å lage et program med grafisk brukergrensesnitt (GUI), hvor brukeren kan velge kommune og få oppgitt befolkningen. Vi bruker `ttk`-modulen for å få moderne *widgets*. I dette eksempelet bruker vi `place` for å plassere dem med `x`- og `y`-

posisjoner. Vi fyller `Combobox`-ens `values`-parameter med alle kommunene og kobler hendelsen `<<ComboboxSelected>>` til funksjonen `vis_befolkning`. Når brukeren har valgt en kommune, slår vi opp kommunen (`e.widget.get()`), finner `commune_nr`, henter data for befolkningen (via en POST API-spørring) og viser det i en `Label`. Funksjonen `sjekk_input` er koblet til hendelsen `<KeyRelease>` for å kunne filtrere valgmulighetene i `Combobox`-en. `vis_befolkning()` og `sjekk_input()` er definert som indre funksjoner innenfor `main()`, fordi de ikke brukes andre steder i programmet og for å unngå bruk av globale variabler. Alternativt kunne vi ha løst dette ved å overføre flere argumenter til funksjonene.

```
def sjekk_input(event):
    value = event.widget.get()
    combo["values"] = [item for item in kommuner if value.lower() in item.lower()]

def vis_befolkning(event):
    selected_kommune = event.widget.get()
    commune_nr = str(df[df["kommune"] == selected_kommune]["kommune_nr"].iloc[0]).zfill(4)
    result = hent_befolkning(f"K-{commune_nr}")
    if result:
        tekst_ut.config(text=f"{result['kommune']} hadde {result['befolkning']}").replace(",", " ")
    else:
        tekst_ut.config(text="Data ikke tilgjengelig")

combo.bind("<KeyRelease>", sjekk_input)
combo.bind("<<ComboboxSelected>>", vis_befolkning)
```

I denne første utgaven har vi valgt en prosedyreorientert tilnærming. Programmet ligger i mappen `b_4_09_1_SSB/1_POP`, sammen med dokumentasjonen i `/docs`.



## befolkning

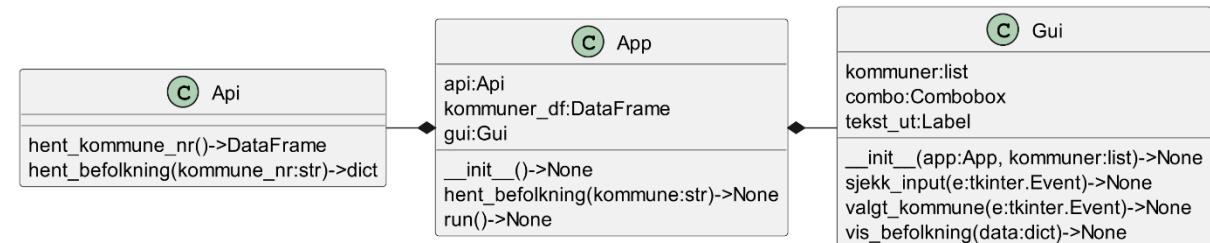
Vis befolkningen til en valgfri kommune med API-kall til Statistisk sentralbyrå (SSB)

Funksjoner:

- `hent_kommune_nr` - Henter kommuneinndeling fra SSB og returnerer en DataFrame.
- `hent_befolkning` - Henter befolkningstall for en spesifikk kommune.
- `main` - Initialiserer og konfigurerer GUI-komponentene.

[▶ View Source](#)

En objektorientert versjon med tilsvarende dokumentasjon ligger i mappen `b_4_09_1_SSB/2_OOP`.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### API Documentation

```
class App
    App()
    api
    kommuner_df
    gui
    hent_befolkning()
    run()
class Api
    hent_kommune_nr()
    hent_befolkning()
class Gui
    Gui()
    app
```

## befolkning

Bruk SSB og vis befolkningen til en valgfri kommune.

### Klasser:

- App - Håndterer samspillet mellom Api og Gui.
- Api - Håndterer API-kall til SSB for å hente kommuneinndeling og befolkningstall.
- Gui - Håndterer det grafiske brukergrensesnittet.

[▶ View Source](#)

### class App:

Applikasjonens hovedklasse som håndterer API-kall og GUI.

[▶ View Source](#)

### Metoder:

- `__init__` - Initialiserer App-klassen ved å opprette en Api- og Gui-instans.
- `hent_befolkning` - Henter befolkningstallet for en valgt kommune.
- `run` - Starter GUI-hendelsessløyfen.

Til slutt har vi tatt med en versjon hvor vi har beholdt `App`- og `Gui`-klassene, men lagt metodene i `Api`-klassen ut i en modul `api.py` med funksjonene `hent_kommune_nr` og `hent_befolkning`. Se mappe `b_4_9_1_SSB/3_OOP_med_POP_modul`.

### Available Modules

```
befolkning
api
```



Search API Documentation...

Vi venter med å vurdere de forskjellige versjonene til øvingsoppgavene.

### REST-API

Vi brukte SSBs veiledning uten å forklare hvilken teknologi som benyttes og hva som foregår. Det skal vi se litt nærmere på her.

**API** (*Application Programming Interface*) er en måte å kommunisere mellom to programmer, enten det er på samme enhet eller over nettverk. Dokumentasjonen viser hvilke funksjoner og metoder som er tilgjengelige, hvordan de skal brukes, datatyper, parametere, autentisering, og hva slags returverdier man kan forvente, med mer. API kan være uavhengig både av HTTP og Internett og kan benyttes i et intranett, LAN eller til kommunikasjon mellom programmer på samme datamaskin.

Vi kan hente data fra skyen på flere måter. HTTPS protokollen brukes for det meste til å laste ned websider, SFTP protokollen til å laste ned filer, IMAP til e-postmeldinger, osv.

**HTTPS** inneholder metoder som **GET** for å hente data, **POST** for å sende data, **PUT** for å oppdatere data, og **DELETE** for å slette data. Statistisk sentralbyrå bruker POST når vi skal hente data. Dette kan være for å øke sikkerheten. Når vi bruker GET-metoden, som i eksempelet her for å hente data fra [Brønnøysundregistrene](#), sendes argumentene med i URL-en og lagres i nettleseres historikk.

<https://data.brreg.no/rofs/od/rofs/stottetildeling/search?language=nob&mottakerOrgnr=987&fraDato=2022-01-01>

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Når vi bruker POST-metoden, kan vi sende argumentene i selve meldingen. Eventuelle sensitive data som passord, blir dermed ikke eksponert i URL-en. For å sikre at dataene er kryptert, bør vi bruke HTTPS i stedet for HTTP.

**REST** (*Representational State Transfer*) er en arkitekturstandard for utvikling av nettverkstjenester, som definerer regler for hvordan tjenester skal bygges og kommunisere. Det bruker vanligvis HTTPS for kommunikasjon og dataene er representert i form av JSON eller XML. REST er ingen offisiell standard, men en "de facto" standard. Det vil si at den brukes i praksis av mange utviklere og organisasjoner, selv om den ikke er godkjent av en standardiseringsorganisasjon. Arkitekturprinsippene er hentet fra [doktoravhandlingen](#) til Roy Fielding, som han publiserte i 2000.

**REST-API** er et API som benytter REST-arkitekturen. REST er teorien og REST-API er implementeringen. Da vi hentet data fra Statistisk sentralbyrå, brukte vi SSB sitt REST-API, som igjen benytter HTTPS POST. Vi må ikke forveksle dette API-et med vårt eget API i befolkning-programmet, heller ikke at vi har valgt å kalle en klasse for `Api` (i 2\_OOP) eller en modul for `api` (i 3\_OOP\_med\_POP\_modul).

### Valutakurser (NB)

The screenshot shows the Norges Banks datatorg API documentation page. At the top, there's a header with the Norges Bank logo and a 'Start' button. Below the header, the title 'Norges Banks datatorg' is displayed. A sub-section title 'Norges Banks API for åpne data' follows, with a descriptive text about the REST API implementation. A code snippet in the bottom left shows how to import pandas and read the data from a URL.

I dette eksempelet skal vi hente [valutakurser](#) fra Norges Bank. Da vi hentet befolkningsdata fra SSB, hentet vi alltid dataene fra 1. kvartal 2024. Nå skal vi hente dagens valutakurser hver dag programmet kjører, og vise utviklingen de siste 12 månedene i et linjediagram ved hjelp av `seaborn` og `tkinter`. (Se [Seaborn-diagram i tkinter-vindu](#), bok 3.9 Reelle datasett med OOP og GUI

Vi kan lese [dokumentasjonen](#) eller velge EUR, API og CSV fra [valutakurser](#). URL-en for resultatet kan kopieres fra *popup*-vinduet som åpnes på skjermen. Den kan vi bruke i et Python program og lese dataene direkte inn i

```
1 import pandas as pd
2 url = 'https://data.norges-bank.no/api/data/EXR/B.EUR.NOK.SP?format=csv'
3 valuta_kurser = pd.read_csv(url, sep=";", decimal=",")
4 print(valuta_kurser.head(3))
```

✓ 0:1s

	FREQ	Frekvens	BASE_CUR	Basisvaluta	QUOTE_CUR	Kvoteringsvaluta	TENOR	\
0	B	Virkedag	EUR	Euro	NOK	Norske kroner	SP	
1	B	Virkedag	EUR	Euro	NOK	Norske kroner	SP	
2	B	Virkedag	EUR	Euro	NOK	Norske kroner	SP	

Løpetid DECIMALS CALCULATED UNIT\_MULT Multiplikator COLLECTION \

0	Spot	4	False	0	Enheter	C	
1	Spot	4	False	0	Enheter	C	
2	Spot	4	False	0	Enheter	C	

Innsamlingstidspunkt TIME\_PERIOD OBS\_VALUE

0	ECB concertation tidspunkt	14:15 CET	2022-01-31	10.0085
1	ECB concertation tidspunkt	14:15 CET	2022-02-01	9.9638
2	ECB concertation tidspunkt	14:15 CET	2022-02-02	9.9228

### Oppsett for API

#### Format

- XML (SDMX versjon 2.1)
- JSON (SDMX JSON)
- CSV (Kommaseparert med ID og tittel)
- XLSX (Med ID og tittel)

#### Tidsperiode

- Valgt tidsperiode (2022-01-29 - 2023-01-29)
- Dynamisk (Velg antall perioder du ønsker å få tilbake med utgangspunkt i nyeste observasjon)

#### URL for resultat

https://data.norges-bank.no/api/data/EXR/B.EUR.NOK.SP?format=csv&startPeriod=2022-01-29&endPe

Vurderingsseksempler

en `pandas DataFrame`.

Programmet kan gjøres om til en mer generell funksjon hvor vi bruker valutakode (BASE\_CUR) som parameter. URL-en kan vi gjøre om til en *f-string* så vi kan bytte ut

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

valutakoden og datoene med innholdet fra variabler. Vi lar programmet velge dagens dato og ett år tilbake i tid.

```
url="https://data.norges-bank.no/api/data/EXR/B.EUR.NOK.SP?  
format=csv&startPeriod=2022-01-29&endPeriod=2023-01-29&locale=no&bom=include"
```

```
url = f"https://data.norges-bank.no/api/data/EXR/B.{kode}.NOK.SP?  
format=csv&startPeriod={et_år_siden}&endPeriod={i_dag}&locale=no&bom=include"
```

Vi dropper kolonnene vi ikke har bruk for og døper om kolonnenavnene til **Kode**, **Valuta**, **Dato** og **Kurs**. Dessuten gjør vi om datoverdier fra **str** til **pandas Timestamp** objekter.

```
valuta_koder.txt  
1 USD - Amerikanske dollar  
2 AUD - Australiske dollar  
3 BDT - Bangladeshi taka
```

Valutakodene trenger vi når vi skal velge valuta fra en *Combobox*. De var ikke lett tilgjengelige, så vi kopierte dem fra rullgardinlisten til Norges Bank sin *Combobox* og lagret verdiene i filen **valutakoder.txt**.

Linjene i filen leses lett inn i et **list** objekt som vi kan bruke i *Comboboksen* vår. Vi har nå laget de funksjonene vi trenger for å vise dataene i et grafisk brukergrensesnitt. Vårt API består av to funksjoner, **hent\_valuta\_kurser** og **hent\_valuta\_koder**. **hent\_valuta\_kurser** bruker

Norges Bank sitt REST-API til å hente det siste årets valutakurser for en valgfri valuta. **hent\_valuta\_koder** returnerer en liste med valutakoder og navn.

Med tanke på de foregående eksemplene, bør koden være forståelig. Hvis vi ønsker at et diagram skal vises når vi starter programmet, må denne koden ligge i konstruktøren **\_\_init\_\_**. Det er den samme koden som vi kjører i lytterfunksjonen **vis\_graf**, med ett unntak. *Comboboxen* sender med hendelsesobjektet som argument når brukeren har valgt en valuta. Vi kan likevel kalle denne funksjonen fra **\_\_init\_\_** hvis vi oppretter et hendelsesobjekt **tk.Event()** selv, i **self.vis\_graf(tk.Event())**.

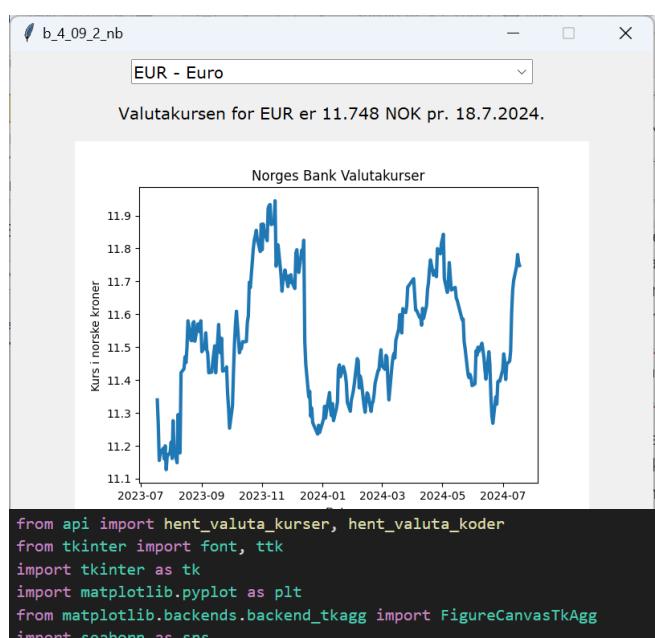
Hvordan får vi det første diagrammet til å vise Euro? Vi har satt *Comboboxen* til stå

```
def hent_valuta_kurser(kode: str) -> pd.DataFrame:  
    """Hent valutakurser for valutaen med valutakoden kode."""  
    i_dag = date.today()  
    et_år_siden = i_dag.replace(year=i_dag.year - 1)  
    url = f"https://data.norges-bank.no/api/data/EXR/B.{kode}.NOK"  
    valuta_kurser = pd.read_csv(url, sep=";", decimal=",")  
    valuta_kurser = valuta_kurser[["BASE_CUR", "Basisvaluta", "TIME_PERIOD", "OBS_VALUE"]]  
    valuta_kurser.columns = ["Kode", "Valuta", "Dato", "Kurs"]  
    valuta_kurser["Dato"] = pd.to_datetime(valuta_kurser["Dato"], format="%Y-%m-%d")  
    return valuta_kurser  
  
print(hent_valuta_kurser("EUR").head(3))  
  
  ✓ 3.8s  


| Kode | Valuta | Dato | Kurs       |         |
|------|--------|------|------------|---------|
| 0    | EUR    | Euro | 2023-07-18 | 11.3395 |
| 1    | EUR    | Euro | 2023-07-19 | 11.2660 |
| 2    | EUR    | Euro | 2023-07-20 | 11.1555 |


```

```
from pathlib import Path  
  
def hent_valuta_koder() -> list:  
    """Hent alle valutakodene som Norges Bank har kurs for."""  
    fil = Path(__file__).parent.resolve().joinpath("valuta_koder.txt")  
    with open(fil, encoding="utf-8") as f:  
        return list(f.read().splitlines())  
  
print("\n".join(hent_valuta_koder()[0:3]))  
USD - Amerikanske dollar  
AUD - Australiske dollar  
BDT - Bangladeshi taka
```



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

på Euro når vi starter:

`self.combo.current(7)`. Dessuten henter vi ikke lenger valutakoden fra hendelsesobjektet med `e.widget.get()`, men bruker `self.combo.get()`. Vi bruker `[3]` for å hente ut bare de tre første bokstavene av tekststrengen til koden. Når vi skal vise dagens valutakurs, bruker vi `.iloc[-1]` til å plukke ut den siste raden `self.valuta_kurser (DataFrame)`. Filene ligger i mappen `b_4_9_2_nb`.

```
class Gui(tk.Tk):
    """GUI for å vise valutakurser fra Norges Bank."""

    def __init__(self):
        """Initialiser GUI."""
        super().__init__()
        self.title("b_4_09_2_nb")
        self.geometry("800x600")
        self.resizable(False, False)
        default_font = font.nametofont("TkDefaultFont")
        default_font.configure(family="Verdana", size=15)
        self.option_add("*TCombobox*Listbox.font", default_font)
        self.combo = ttk.Combobox(
            self, values=hent_valuta_koder(), font=default_font, width=36
        )
        self.combo.pack(pady=10)
        self.combo.bind("<><ComboboxSelected>>", self.vis_graf)
        # Start med Euro
        self.combo.current(7)
        self.label = tk.Label(self, text="Dagens kurs")
        self.label.pack(pady=10)
        self.fig, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(figure=self.fig, master=self)
        self.vis_graf(tk.Event())
        self.canvas.get_tk_widget().pack(pady=10)

    def vis_graf(self, e):
        """Vis valutakursen og diagram for valutaen valgt i comboboxen."""
        kode = self.combo.get()[3]
        self.valuta_kurser = hent_valuta_kurser(kode)
        kurs = self.valuta_kurser["Kurs"].iloc[-1]
        dato = self.valuta_kurser["Dato"].iloc[-1]
        dato = f"[dato.day].{dato.month}.{dato.year}"
        self.label.configure(
            text=f"Valutakursen for {kode} er {kurs} NOK pr. {dato}."
        )
        self.ax.clear()
        self.ax = sns.lineplot(
            x="Dato", y="Kurs", data=self.valuta_kurser, linewidth=3
        )
        plt.title("Norges Bank Valutakurser")
        plt.ylabel("Kurs i norske kroner")
        self.canvas.draw()

    def destroy(self):
        plt.close('all')
        super().destroy()
        self.quit()

if __name__ == "__main__":
    gui = Gui()
    gui.mainloop()
```

API Documentation

hent\_valuta\_kurser()  
hent\_valuta\_koder()

built with 

### api

Hent valutakurser fra Norges Bank

Funksjoner:

```
hent_valuta_kurser(kode)
hent_valuta_koder()
```

Kode er valutakoden, f.eks. EUR for euro.

[View Source](#)

```
def hent_valuta_kurser(kode: str) -> pandas.core.frame.DataFrame:
```

[View Source](#)

Hent valutakurser for valutaen med valutakoden kode.

```
def hent_valuta_koder() -> list:
```

[View Source](#)

Hent alle valutakodene som Norges Bank har kurs for.

### Ta med minst dette

Vi har brukt reelle datasett med REST API for å hente befolkningsdata fra Statistisk sentralbyrå (SSB) og valutakurser fra Norges Bank (NB). Disse API-spørringene henter spesifikke deler av datasett, tilsvarende spørrengjer i en database. Vi har også laget programmer med grafisk brukergrensesnitt (GUI). I det ene kan brukeren velge kommune og få oppgitt befolkningen, og i det andre kan brukeren velge valutakurs og se siste års kurser i et linjediagram for denne valutaen.

Vurderingssempolar

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## Øvingsoppgaver

### 4.9.1

Vi lagde tre versjoner av programmet som slo opp befolkningen i valgt kommune:

1. [b\\_3\\_9\\_1\\_SSB/1\\_POP](#)
2. [b\\_3\\_9\\_1\\_SSB/2\\_OOP](#)
3. [b\\_3\\_9\\_1\\_SSB/3\\_OOP\\_med\\_POP\\_modul](#)

Vurder de tre versjonene opp mot hverandre med tanke på fordeler og ulemper med hver enkelt versjon.

### 4.9.2

Lag en applikasjon som viser når neste fly lander på Gardermoen. Bruk [Avinor Flydata](#). For å hente ut flyvninger fra XML dataene, kan vi bruke `df = pd.read_xml(url, xpath='//flight')` samt `pip install -U lxml` terminalvinduet. Oppdater informasjonen hvert 3. minutt og bruk tkinter til å vise

Neste ankomst til Gardermoen      tt.mm.ss. (klokke som oppdateres hvert sekund)

Klokkeslett : \_\_  
Flyvning : \_\_  
Flyselskap : \_\_  
Avgangsflyplass : \_\_

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 4.10 Posisjonsdata

#### Bolkens innhold

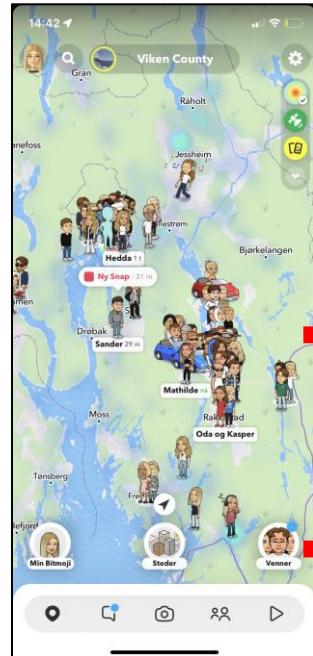
Persondata .....	316
Pesonopplysningsloven og GDPR.....	317
Muligheter, utfordringer og konsekvenser.....	317
Etiske dilemmaer.....	318
Ta med minst dette.....	320
Øvingsoppgaver.....	320

#### Kompetanse mål:

- utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger
- drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn

#### Persondata

Posisjonsdata gir informasjon om en persons eller enhets geografiske plassering til enhver tid og kan samles inn fra forskjellige kilder. Mens geodata gir generell informasjon om geografiske trekk og terrenget, kan posisjonsdata brukes til å spore nøyaktig hvor personen eller enheten har beveget seg. Posisjonsdata kan også inneholde andre dataelementer, for eksempel informasjon om brukeratferd eller brukerpreferanser, avhengig av hvordan dataene er samlet inn og hva som er formålet med å bruke dem. Med utbredelsen av mobile enheter som smarttelefoner og nettbbrett har det blitt mulig å samle inn og analysere store mengder posisjonsdata om bevegelsesmønstre og geografisk plassering av enkeltpersoner og grupper. Andre kilder til posisjonsdata kan være biler, klokker og ulike dappeditter i tingenes internett. Posisjonsdata brukes i stadig større grad av både enkeltpersoner og bedrifter, og det er viktig å forstå hvordan dataene kan brukes på en trygg og ansvarlig måte.



Det går an å kjøpe posisjonsdata av dataforhandlere, og selv om dataene skal være anonymisert, er det likevel mulig å koble informasjonen til en person. I artikkelen [Avslørt av mobilien](#) forteller NRK hvor enkelt de fant ut hvem dataene måtte tilhøre og sporet personens bevegelser i 200 dager, bedre enn vedkommende husket selv. Hvis vi har mobilen hjemme om natten og på jobben om dagen, er det kanskje ikke så vanskelig å finne ut hvem som bor på en bestemt adresse og hvor de jobber?

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Pesonopplysningsloven og GDPR

I 2018 ble [Personopplysningsloven](#) erstattet av en ny lov om behandling av personopplysninger. Denne nye loven bygger på EUs personvernforordning ([GDPR](#)), som har som formål å beskytte personvernet til innbyggere i EU og EØS-området, som inkluderer Norge. Lovens grunnleggende prinsipper gjelder for alle som håndterer personopplysninger. Den behandlingsansvarlige har ansvaret for å overholde disse prinsippene og må kunne dokumentere at de følges:

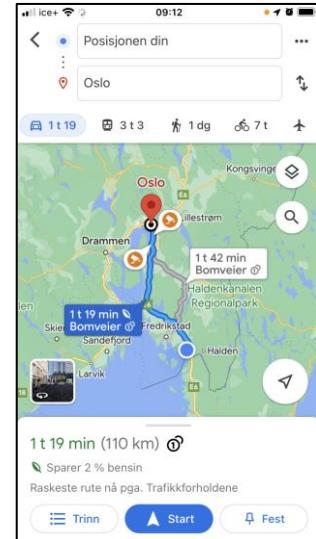
- **Lovlighet, rettferdighet og åpenhet:** Personopplysninger skal behandles på en lovlig, rettferdig og åpen måte med hensyn til de registrerte.
- **Formålsbegrensning:** Personopplysninger skal samles inn og brukes kun til spesifikke, uttrykkelig angitte og berettigede formål, og ikke behandles på en måte som er uforenlig med disse formålene.
- **Dataminimering:** Personopplysninger skal kun samles inn og behandles i den grad det er nødvendig for formålet de skal brukes til.
- **Riktighet:** Personopplysninger skal være korrekte og oppdaterte, og det skal tas tiltak for å rette eller slette uriktige opplysninger.
- **Lagringsbegrensning:** Personopplysninger skal lagres i en begrenset periode og kun så lenge som det er nødvendig for formålene de behandles for.
- **Integritet og konfidensialitet:** Personopplysninger skal behandles på en måte som sikrer tilstrekkelig sikkerhet mot uautorisert eller ulovlig behandling og utilsiktet tap, ødeleggelse eller skade.

### Muligheter, utfordringer og konsekvenser

Posisjonsdata kan være et tvegget sverd, med både muligheter og utfordringer knyttet til bruken av dataene. På den ene siden gir posisjonsdata store muligheter for å analysere bevegelsesmønstre og geografisk plassering av enkeltpersoner og grupper, og dette kan brukes til å utvikle bedre tjenester og produkter, spesielt innen transportsektoren. For enkeltpersoner kan posisjonsdata også gi verdifull informasjon om for eksempel reiseruter og trafikkforhold.

På den andre siden kan posisjonsdata også utgjøre en utfordring for personvernet. Det kan være enkelt å koble informasjonen til en bestemt person, selv om dataene skulle være anonymiserte. Dette kan føre til at personer kan bli overvåket og sporet uten at de er klar over det eller har gitt samtykke til det. Det er derfor viktig å være bevisst på både mulighetene og utfordringene ved bruk av posisjonsdata, og å sørge for at dataene brukes på en trygg og ansvarlig måte.

Så posisjonsdata kan ha både positive og negative konsekvenser, og det er viktig å forstå risikoen ved å bruke denne typen informasjon. Feil bruk av posisjonsdata kan ha alvorlige konsekvenser for enkeltpersoner og samfunnet som helhet. Nedenfor følger noen eksempler på hvordan posisjonsdata kan brukes til å overvåke, diskriminere og utnytte enkeltpersoner, samt føre til sikkerhetsrisikoer og tap av ferdigheter.



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

- **Overvåkning og kontroll:** Posisjonsdata kan brukes til å overvåke og kontrollere mennesker uten deres samtykke eller kunnskap, noe som kan føre til en følelse av mistillit og manglende frihet. Dette kan også være en trussel mot demokratiske verdier, og det er derfor viktig å sørge for at dataene ikke brukes på en måte som begrenser folks frihet.
- **Diskriminering:** Hvis posisjonsdata blir brukt på en diskriminerende måte, for eksempel for å utelukke bestemte grupper fra tilgang til tjenester eller arbeidsplasser, kan det føre til ulikheter og urettferdighet.
- **Sikkerhetsrisikoer** Posisjonsdata kan også føre til sikkerhetsrisikoer hvis de havner i feil hender. Dataene kan brukes til å planlegge kriminelle handlinger eller terrorangrep, og det er derfor viktig å sørge for at dataene er godt beskyttet og at bare de som trenger tilgang til dem, har det.
- **Avhengighet og tap av ferdigheter:** Bruk av teknologi som er avhengig av posisjonsdata, for eksempel navigasjonsapper, kan føre til tap av navigasjonsferdigheter og orienteringsevne over tid. Det kan bli et problem i situasjoner når teknologien ikke er tilgjengelig eller fungerer dårlig. Vi kan miste evnen til å navigere og orientere oss på egen hånd, og vi kan også bli dårligere til å huske reiseruter, planlegge alternative ruter og utforske nye steder.
- **Økonomisk utnyttelse:** Posisjonsdata kan også brukes til økonomisk utnyttelse av enkeltpersoner, for eksempel ved at prisene på produkter eller tjenester settes høyere i områder med høy etterspørsel, eller ved at reklame rettes spesifikt mot enkeltpersoner basert på deres posisjon. Dette kan føre til økonomisk ulikhet og utnyttelse av enkeltpersoner.

#### **Etiske dilemmaer**

Etiske dilemmaer knyttet til posisjonsdata kan oppstå når det er konflikt mellom hensynet til personvern og nytten av å bruke posisjonsdata. Bruken av posisjonsdata kan være svært inngrpende og påvirke personvernet til enkeltpersoner og grupper. Her er noen eksempler på etiske dilemmaer som kan oppstå i forbindelse med bruk av posisjonsdata:

- **Overvåkning og sporing:** Bruk av posisjonsdata kan føre til uønsket overvåkning og sporing uten samtykke, som kan oppleves som en innstrenging på privatliv og personvern. Dette kan skape et etisk dilemma for enkeltpersoner som ønsker å balansere behovet for personlig frihet og beskyttelse mot sikkerhetsrisikoer.
- **Deling av data:** Å dele posisjonsdata med tredjeparter kan føre til personvernbrudd og invasjon av personlig integritet. Det kan skape et etisk dilemma for brukere mellom ønsket om personlig tilpasset innhold og beskyttelse av privatliv.

## Smidig IT-2 for eksklusiv bruk ved <navn på skole>

- **Diskriminering:** Personer kan bli utelukket fra jobbmuligheter på grunn av sin posisjonshistorikk, som kan vise at de har oppholdt seg på utesteder for LHBTQ+. Dette kan føre til at velkvalifiserte jobsøkere ikke får jobben på grunn av sin seksuelle orientering og et etisk dilemma for arbeidsgiveren, som i så fall også handler i strid med [likestillings- og diskrimineringsloven](#).
- **Feilaktige beslutninger:** Personer kan få dårligere forsikringsdekning eller høyere priser på grunn av sin posisjonshistorikk. Selv om den viser at de har oppholdt seg på steder som ansees for å ha høy risiko, behøver det ikke være noen økt risiko for skade eller tap i fremtiden. Dette kan føre et etisk dilemma for forsikringsselskapet, som må balansere behovet for å maksimere fortjeneste med ansvarlig og rettferdig behandling av kundene.
- **Sikkerhetsrisiko:** Bruk av posisjonsdata kan også utgjøre en sikkerhetsrisiko. Hackere kan få tilgang til sensitiv informasjon basert på posisjonsdata, som kan føre til tap av personopplysninger og identitetstyveri. Her vil tjenesteleverandøren stå overfor et etisk dilemma når det gjelder å balansere fordelene ved å samle inn og bruke posisjonsdata mot risikoen for sikkerhetsbrudd og tap av personopplysninger.



For å unngå etiske dilemmaer knyttet til posisjonsdata, er det viktig å være bevisst på hvorfor vi samler inn dataene og å gjøre det tydelig for brukerne hva slags informasjon som blir samlet inn og hvordan den vil bli brukt. Det er også viktig å samle inn dataene på en trygg og ansvarlig måte, og beskytte personopplysningene mot uautorisert tilgang og misbruk. Videre bør vi gi brukerne muligheten til å bestemme om de vil dele posisjonsdataene sine eller ikke, og slette dem hvis de ønsker det, slik at vi ivaretar personvernet og integriteten til brukerne.

Vi minner om at etikk handler om å reflektere over spørsmålet om hva som er rett og galt. Når vi drøfter etiske dilemmaer, er det viktig å være bevisst på hvilke ord og uttrykk vi bruker. Her er noen nyttige uttrykk: "For det første, for det andre", "I motsetning til", "Til forskjell fra", "På den ene siden, og på den andre siden" og "Men". Vi kan også ta i bruk ulike tilnærminger når vi diskuterer etiske dilemmaer: konsekvensetikk, pliktetikk og holdningsetikk.

Konsekvensetikk innebærer å vurdere konsekvensene av våre handlinger, både for de direkte involverte og resten av menneskeheden, før vi tar en beslutning. Pliktetikk handler om å følge lover, regler og normer som er etablert for å sikre etisk handling. Holdningsetikk innebærer å handle etter ens egen overbevisning og "hjerte", uten å ta hensyn verken til konsekvenser eller plikt.

Det er viktig å være oppmerksom på at konsekvensetikken kan ha en svakhet, siden det kan være umulig å overskue alle konsekvensene av våre handlinger. Hvis vi forventer å ha full kontroll, kan vi i prinsippet aldri handle. Til syvende og sist er mange av våre valg og beslutninger basert på våre holdninger og verdier. Det er også viktig å være bevisst på signaleffekten disse valgene har på andre.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Ta med minst dette

Posisjonsdata gir detaljert geografisk informasjon om bevegelser og oppholdssteder for personer og enheter. Denne informasjonen byr på både muligheter og utfordringer. På den ene siden kan dataene forbedre tjenester og produkter, som for eksempel innen transportsektoren. På den andre siden reiser de alvorlige personverns- og sikkerhetsspørsmål. Bruken av posisjonsdata krever derfor nøye overveielse av etiske dilemmaer, spesielt relatert til personvern, sikkerhet og mulig diskriminering. Personopplysningsloven og GDPR setter rammer for håndtering av persondata, men i praksis er det ikke alltid så rett frem. Det er mange gråsoner og vanskelige spørsmål som dukker opp.

### Øvingsoppgaver

De to først oppgavene er hentet fra UDIRs [eksempeloffgangsoppgave for eksamen i IT-2](#), som kom høsten 2022.

### Informasjonsside om Posisjonsdata og mobilsporing

Denne seksjonen inneholder to oppgaver. Du skal bruke artiklene fra forberedelsdelen og annen kunnskap og kilder du har til å løse disse to oppgavene. Du bør sette av inntil 30 minutter til disse to oppgavene, oppgave 12 vil kreve mest tid av disse.

I 2019 gjennomførte NRK Beta et kjøp av posisjonsdata fra et britisk selskap og undersøkte [hva dette kunne brukes til](#). Dette ble starten på et [journalistisk prosjekt](#) om mobilsporing.

#### 4.10.1 Posisjonsdata

Ta utgangspunkt i din lesing av artikkelen om Posisjonsdata og mobilsporing og avgjør om følgende påstander er sanne eller usanne:

	sant	usant
Posisjonsdata fra mobiler er alltid anonymiserte	<input type="checkbox"/>	<input type="checkbox"/>
Posisjonsdata blir registrert ved hjelp av en GPS-sender på mobilen	<input type="checkbox"/>	<input type="checkbox"/>
Når vi gir apper tilgang til posisjonsdata for mobilen vår, kan de samle inn dataene og selge de videre	<input type="checkbox"/>	<input type="checkbox"/>
Posisjonsdata kan selges videre kun til myndigheter	<input type="checkbox"/>	<input type="checkbox"/>
Å selge posisjonsdata er i strid med personvernloven	<input type="checkbox"/>	<input type="checkbox"/>
Det eneste som hjelper mot at posisjonsdataene våre kan bli solgt videre, er å la være å installere apper	<input type="checkbox"/>	<input type="checkbox"/>
Selskaper kan samle inn personopplysninger som navn og posisjonsdata så lenge de har en grunn til det	<input type="checkbox"/>	<input type="checkbox"/>

#### 4.10.2 Drøftingsoppgave om posisjonsdata

Skriv en kort tekst om innsamling av posisjonsdata fra apper på mobiltelefoner. Teksten skal vurdere muligheter og utfordringer ved bruk av apper som samler slike data og drøfte konsekvenser av dette for individ og samfunn.

#### 4.10.3 Etisk dilemma

Drøft et av de gitte eksemplene på etiske dilemmaer i teksten med fokus på konsekvensetikk, pliktetikk og holdningsetikk.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 4.11 Oppsummering

*utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger*

Vi har drøftet muligheter og utfordringer knyttet til bruk av posisjonsdata, samt konsekvensene av feil bruk av slik informasjon. Videre har vi diskutert konsekvensene av god og dårlig brukervennlighet i IT-systemer. Vi har også utforsket og vurdert muligheter og begrensninger med verktøy som *Live Share* i *VS Code*, *GitHub*, *Trello* og *Slack* i systemutvikling.

*drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn*

Posisjonsdata kan gi opphav til etiske dilemmaer når det oppstår en konflikt mellom personvern og nytten av å bruke slike data. Dette inkluderer overvåkning og sporing uten samtykke, deling av data med tredjeparter, diskriminering basert på posisjonshistorikk, feilaktige beslutninger og sikkerhetsrisikoer. For å håndtere disse dilemmaene er det viktig å være bevisst på formålet med datainnsamlingen, informere brukerne tydelig, sikre trygg håndtering av dataene og gi brukerne kontroll over deling og sletting av posisjonsdataene. Når vi drøfter etiske dilemmaer, kan ulike tilnærmingar som konsekvensetikk, pliktetikk og holdningsetikk brukes til å reflektere over hva som er rett og galt.

*utforske og vurdere alternative løsninger for design og implementering av et program*

Vi har utviklet programmer, både prosedyreorientert og objektorientert, samt vurdert deres fordeler og ulemper, og vi har vurdert alternative løsninger for tidsstyrte hendelser, med mer.

*anvende objektorientert modellering til å beskrive et programs struktur*

Malen for Pygame-programmet som presenteres i teksten, er basert på objektorientert modellering og hjelper oss til å forstå hvordan klasser og objekter brukes til å organisere og strukturere programmet. Vi har også utforsket bruksmønstre, klassediagrammer og sekvensdiagrammer for å beskrive systemets funksjonalitet og struktur.

*utvikle objektorienterte programmer med klasser, objekter, metoder og arv*

Eksemplene med Pygame viser hvordan vi kan utvikle objektorienterte programmer ved å opprette klasser, objekter, metoder og bruke arv. Vi har utviklet flere objektorienterte simulatingsprogrammer, og etter at vi lærte mer om modellering, implementerte vi disse prinsippene i praktiske eksempler for å se hvordan de fungerer i virkelige programmer.

*vurdere og bruke strategier for feilsøking og testing av programkode*

Vi har utforsket integrasjonstesting og GUI-testing og lært å organisere koden for effektiv testing, som også fremmer gjenbruk og gjør koden lettere å lese. Integrasjonstesting sikrer at ulike enheter fungerer sammen som forventet. I GUI-testing skiller vi brukergrensesnittet fra programmets logikk for å kunne teste delene hver for seg. Automatisering av GUI-tester kan effektivisere prosessen, selv om noen manuelle tester fortsatt er nødvendige for å sikre en god brukeropplevelse.

Vurderingsksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

*generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode*

I eksemplene og oppgavene med Pygame har vi utviklet metoder som bidrar til å generalisere løsninger. Videre har vi benyttet *pandas*-biblioteket for å håndtere og analysere data, noe som innebærer bruk av gjenbrukbar programkode.

*vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer*

Vi har gitt en innføring i definisjonen av brukervennlighet, presentert kriterier for vurdering, og foreslått metoder for å forbedre brukergrensesnittet. Videre har vi lært hvordan vi kan gjøre programmene mer interaktive ved å håndtere hendelser. Gjennom kollisjoner blir spill og applikasjoner mer engasjerende, med tidsstyring kan vi skape dynamiske opplevelser, og GUI-integrering med Pygame muliggjør et mer intuitivt grensesnitt..

*velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre*

Det er viktig å kunne velge og bruke riktig verktøy til riktig formål. Vi har lært å bruke verktøy som *Live Share* i *VS Code*, *GitHub*, *Trello* og *Slack*, som har latt oss kunne jobbe sammen i sanntid, dele kode, administrere et prosjekt og kommunisere effektivt.

*gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data*

Vi har beskrevet standarder for lagring, utveksling og sikring av ulike typer data, inkludert karakterkodingsstandarder, filformater, kommunikasjonsprotokoller og sikkerhetsstandarder. Dessuten har vi belyst betydningen av standarder i forbindelse med kontinuerlig tilgang til eldre data, behovet for konvertering til nye formater og lagringsmedier, samt digitaliseringen av eksponentielt økende nye data.

*bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett*

Vi har jobbet videre med *pandas*-biblioteket for å håndtere og analysere reelle datasett. Dessuten har vi lært å rense og utforske reelle datasett for å sikre kvaliteten i vår analyse og presentasjon. Videre har vi hentet reelle data via REST API-er, som fra SSB og Norges Bank, og presentere disse i grafiske brukergrensesnitt.

## Del 5

### 5.1 Dato og tid

#### Innhold

Innledning .....	323
Dato .....	323
Språkinnstilling (locale) .....	324
Tid.....	324
Dato og tid.....	324
pandas og tid .....	325

#### Innledning

I dag er det lørdag 27. mai 2023. Hvorfor er det viktig å lære om dato og tid i Python? Vel, det er fordi det ikke er så enkelt å utføre operasjoner med teksten "lørdag 27. mai 2023". Hvordan skal et program kunne finne ut hvilken ukedag det er om fire dager, eller hvor mange dager som har gått siden skolestarten den 17. august 2022? Derfor har Python egne klasser for dato, tid og dato og tid. Disse klassene inneholder metoder som lar oss utføre slike operasjoner. Men det er flere utfordringer underveis.

Når en amerikaner skriver 2/3/23, betyr det 3. februar 2023, men i Norge betyr det 2. mars 2023. I tillegg tillater vi også skrivemåter som 2.3.2023, 02.03.2023, 2.3.23 og 02.03.23. Tilsvarende er "4 pm" i USA det samme som klokken 16:00 her i Norge, med unntak av eventuelle tidsforskjeller og sommertid. For å håndtere disse forskjellene, må vi angi hvilken tidssone det gjelder, for eksempel WEST (Western European Summer Time) som er UTC+01 (Coordinated Universal Time). Det er også viktig å være oppmerksom på at forskjellige land og kulturer kan følge ulike kalendersystemer, noe som ytterligere kompliserer dette. Så det spiller faktisk en rolle hvor programmet blir kjørt og hvilke standardinnstillinger som Python bruker.

#### Dato

Vi kan enkelt hente dagens dato med `today`-metoden i `date`-klassen i `datetime`-modulen. `date`-klassen har metoder for å håndtere datoer uten å ta hensyn til tid. Vær oppmerksom på at verdiene hentes fra datamaskinens systemklokke og innstillinger, som kan være feil.

Standardinnstillingene til `date`-objektet er ÅÅÅÅ-MM-DD, i samsvar med [ISO 8601-standarden](#). Vi kan nå finne ut hvilken dato det er i

```
from datetime import date
i_dag = date.today()
print(f"I dag er det {i_dag} ({type(i_dag)})")

I dag er det 2023-05-28 (<class 'datetime.date'>).

from datetime import timedelta
i_morgen = i_morgen = i_dag + timedelta(days=1)
print(f"I morgen er det {i_morgen} ({type(i_morgen)})")

I morgen er det 2023-05-29 (<class 'datetime.date'>).

print(f"År: {i_dag.year} ({type(i_dag.year)})")
print(f"Måned: {i_dag.month} ({type(i_dag.month)})")
print(f"Dag: {i_dag.day} ({type(i_dag.day)})")

År: 2023 (<class 'int'>)
Måned: 5 (<class 'int'>)
Dag: 28 (<class 'int'>)
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

morgen ved bruk av `timedelta`-klassen. Begge datoene får datatypen `datetime.date`. Vi kan hente ut årstall, måned og dag hver for seg, og disse verdiene får datatypen `int`.

Det er også mulig å lage `date`-objekter ved å oppgi år, måned og dag, eller ved å konvertere tekststrenger.

Når vi konverterer tekststrenger med `datetime.strptime`-metoden må vi bruke to argumenter: tekststrengen og formatet. Dessuten må vi sette riktig språkinnstilling ved hjelp av `locale`-modulen for at

programmet skal forstå og bruke norske navn på dager og måneder. Tilsvarende formatering kan brukes med `strftime`-metoden som konverterer `datetime`-objektet til en tekststreng som kan skrives ut. Vi ser også at det er mulig å lese inn amerikansk datoformat slik som det er ment, ved å oppgi korrekt format.

```
skolestart = date(2022, 8, 17)
print(f"Skolestart var {skolestart} ({type(skolestart)}).")
✓ 0.0s
Skolestart var 2022-08-17 (<class 'datetime.date'>).

from datetime import datetime
import locale
locale.setlocale(locale.LC_ALL, "nb_NO")
i_dag = datetime.strptime("lørdag 27. mai 2023", "%A %d. %B %Y").date()
print(f"I dag er det {i_dag} ({type(i_dag)}).")
formateret_dato = i_dag.strftime("%A %d. %B %Y").capitalize()
print(f'{formateret_dato} hadde det gått {(i_dag-skolestart).days} dager siden skolestart.')
✓ 0.0s
I dag er det 2023-05-27 (<class 'datetime.date'>).
Lørdag 27. mai 2023 hadde det gått 283 dager siden skolestart.

amerikansk_dato = datetime.strptime("02/03/23", "%m/%d/%y")
print(f"Amerikansk dato: {amerikansk_dato} ({type(amerikansk_dato)}).")
formateret_dato = amerikansk_dato.strftime("%#d. %B %Y")
print(f"Formatert dato: {formateret_dato} ({type(formateret_dato)}).")
✓ 0.0s
Amerikansk dato: 2023-02-03 00:00:00 (<class 'datetime.datetime'>).
Formatert dato: 3. februar 2023 (<class 'str'>).
```

### Språkinnstilling (locale)

Ved oppstart bruker Python C `locale` uavhengig språkinnstillingene, noe som medfører standardiserte engelske navn. Vi bør unngå `setlocale` i biblioteksrutiner da det påvirker hele programmet som bruker det.

```
import locale
from datetime import date
default_locale = locale.getdefaultlocale()
print(f"Språkinnstilling: {locale.getdefaultlocale()}")
julaften = date(2023, 12, 24)
print(f"Julaften er {julaften:%A %d. %B %Y}.")
locale.setlocale(locale.LC_ALL, "nb_NO")
print(f"Julaften er {julaften:%A %d. %B %Y}.")
✓ 0.0s
Språkinnstilling: ('nb_NO', 'cp1252')
Julaften er Sunday 24. December 2023.
Julaften er søndag 24. desember 2023.
```

### Tid

Med tid mener vi klokkeslett uavhengig dato. `time`-klassen i `datetime`-modulen har begrenset funksjonalitet. For eksempel kan vi ikke beregne differansen mellom to `datetime.time`-objekter. Det finnes imidlertid en annen modul, `time`, der `time`-objekter er UNIX-tid, dvs antall sekunder siden 1. januar 1970 kl 00:00:00 UTC. Disse objektene inneholder dermed både dato- og tidsinformasjon, så vi kan like gjerne bruke `datetime`-objekter fra `datetime`-modulen.

### Dato og tid

På samme måte som vi hentet ut år, måned og dag fra et `date`-objekt kan vi også hente ut timer, minutter, sekunder og mikrosekunder fra et `datetime`-objekt.

```
from datetime import datetime
nå = datetime.now()
print(f"nå: {nå}, ({type(nå)})")
timer, minutter, sekunder = nå.hour, nå.minute, nå.second
print(f"timer: {timer}, ({type(timer)})")
print(f"minutter: {minutter}, ({type(minutter)})")
print(f"sekunder: {sekunder}, ({type(sekunder)})")
✓ 0.0s
nå: 2023-05-28 15:48:30.121960, (<class 'datetime.datetime'>)
timer: 15, (<class 'int'>)
minutter: 48, (<class 'int'>)
sekunder: 30, (<class 'int'>)
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

Som nevnt kan vi bruke `datetime`-objekter til å beregne tidsdifferanser. For eksempel kan vi finne ut hvor lang tid det tar å summere en million vilkårlige tall mellom 1 og 100. Legg merke til at differansen mellom de to `datetime`-objektene blir et `timedelta`-objekt. Vi kan bare hente ut dager, sekunder og mikrosekunder direkte.

Resten av konverteringen må vi gjøre selv.

Et tidspunkt tilhører en tidssone. Derfor må vi oppgi tidssonen når vi angir riktig tid. Når vi oppretter et `datetime`-objekt i Python uten å angi tidssone, blir tidssonen satt til `None`.

`pytz`-modulen forenkler programmering med tidssoner. Både `now`-metoden og `datetime`-konstruktøren kan ta en tidssone (`tzinfo`) som argument. Dermed er det enkelt å vise hva klokka er andre steder i verden. UTC (*Coordinated Universal Time*) er en global standard for tidsangivelser som brukes i alle land.

Det erstattet *Greenwich Mean Time* (GMT) som referanse-tidsskala. UTC er viktig for å koordinere og synkronisere tid over hele kloden og er basert på gjennomsnittet av flere atomur rundt om i verden.

### pandas og tid

`pandas` har en `Timestamp`-klasse som tilsvarer `datetime`, og de kan stort sett brukes om hverandre. Når vi oppretter en `DataFrame` med konstruktøren, `read_csv` eller `read_json`, forblir datoene som tekstrengjer. Vi kan konvertere tekst til datoer med `pandas.to_datetime`-metoden. IT-2 eksamen våren 2023 inkluderte datasettet `googleplaystore.csv`. Det inneholder kolonnen "Last Updated" med datoer som tekst, for eksempel "January 7, 2018". Formatet er satt til "%B %d, %Y". Vi kunne brukt 'mixed', men det er mer risikabelt. Ved å bruke argumentet `errors = 'coerce'` vil verdier som ikke kan konverteres, bli satt til `NaT` (*Not a Time*), som håndteres ved å bruke `isna()` på lik linje med `NaN`. Disse verdiene kan uteslås med `dropna`-metoden (Det er én av dem).

Deretter bytter vi til norsk språkinnstilling. Hvis vi hadde gjort dette før vi leste inn de engelske månedene, ville det ha blitt feil. Vi skriver deretter ut ukedagene på norsk på ulike måter for de fem første oppføringene. Variabelen `date` inneholder dato-en, `day_of_week` inneholder nummeret på ukedagen, `weekday_name` henter navnet på dagen ved å bruke `day_name()`-metoden i `calendar`-modulen. Først skriver vi ut navnet på dagen ved å formtere utskriften til dato-en. (%A). Deretter skriver vi ut nummeret på ukedagen og navnet, som nå er

```
from datetime import datetime
from random import randint

liste = [randint(1, 100) for _ in range(1000000)]
start = datetime.now()
sum_ = sum(liste)
slutt = datetime.now()
varighet = slutt - start
print(f'Sum: {sum_}')
print(f'Det tok {varighet} å summere 1 million tall ({type(varighet)})')
ms = varighet.microseconds/1000
print(f'Det tok {ms} millisekunder å summere 1 million tall ({type(ms)})')

✓ 0.8s
Sum: 50511839
Det tok 0:00:00.007997 å summere 1 million tall (<class 'datetime.timedelta'>)
Det tok 7.997 millisekunder å summere 1 million tall (<class 'float'>)
```

```
nå = datetime.now()
print(f'Tidssonen er satt til {nå.tzinfo}.')
✓ 0.0s
Tidssonen er satt til None.
```

```
from datetime import datetime
import pytz

oslo_tid = datetime.now(pytz.timezone('Europe/Oslo'))
print(f'Tidssonen er satt til {oslo_tid.tzinfo}')
print(f'I Oslo er klokka nå: {oslo_tid.strftime("%H:%M")}')
utc_tid = oslo_tid.astimezone(pytz.utc)
print(f'I UTC er klokka nå: {utc_tid.strftime("%H:%M")}')
new_york_tid = utc_tid.astimezone(pytz.timezone('America/New_York'))
print(f'I New York er klokka nå: {new_york_tid.strftime("%H:%M")}')
✓ 0.0s
Tidssonen er satt til Europe/Oslo
I Oslo er klokka nå: 10:36
I UTC er klokka nå: 08:36
I New York er klokka nå: 04:36
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

hentet fra kalenderen. Til slutt skriver vi ut datoene på standard format. I kalenderen under kan vi se at 7. januar, 2018 virkelig var en søndag.

```
import pandas as pd
import locale
import calendar

df = pd.read_csv("googleplaystore.csv")
print(df["Last Updated"].dtype, type(df["Last Updated"].loc[0]))
df["Last Updated"] = pd.to_datetime(
    df["Last Updated"], format="%B %d, %Y", errors="coerce")
df = df.dropna(subset=["Last Updated"])
print(df["Last Updated"].dtype, type(df["Last Updated"].loc[0]))
locale.setlocale(locale.LC_ALL, "nb_NO")
for _ in range(5):
    date_ = df["Last Updated"].loc[_]
    day_of_week = date_.dayofweek
    weekday_name = calendar.day_name[day_of_week]
    print(f"{date_:%A}, ukedag: {day_of_week}, {weekday_name}, {date_:%d. %B %Y}")

✓ 0.8s
object <class 'str'>
datetime64[ns] <class 'pandas._libs.tslibs.timestamps.Timestamp'>
søndag, ukedag: 0, søndag, 07. januar 2018
mandag, ukedag: 0, mandag, 15. januar 2018
onsdag, ukedag: 2, onsdag, 01. august 2018
fredag, ukedag: 4, fredag, 08. juni 2018
onsdag, ukedag: 2, onsdag, 20. juni 2018
```

januar 2018							^	▼
ma.	ti.	on.	to.	fr.	lø.	sø.		
1	2	3	4	5	6	7		
8	9	10	11	12	13	14		
15	16	17	18	19	20	21		
22	23	24	25	26	27	28		
29	30	31	1	2	3	4		

Når vi har datoer eller tidsstempel i en `DataFrame`, åpner det opp for mange muligheter for analyser og manipulasjoner. Vi kan filtrere rader basert på datoer, gruppere data etter tidspunkt, beregne tidsdifferanser og mye mer.

Samtidig må vi huske at håndtering av dato og tid kan være komplikert på grunn av forskjellige formater, tidssoner og kalendersystemer. Så er det viktig å være nøyne med hvilke biblioteker og moduler vi bruker og å ha god forståelse for de ulike funksjonene og metodene som er tilgjengelige.

Vurderingsseksempler

## 5.2 Tilfeldige tall og utvalg

### Innhold

Innledning .....	327
randint(a,b).....	327
randrange(start, stop, step) .....	327
choice(seq).....	328
choices(seq, k=n).....	328
sample(population, k) .....	328
shuffle(x).....	329
random() .....	329
returner et tilfeldig desimaltall, t, mellom a og b, hvor a <= t < b.....	329
seed(x).....	330

### Innledning

Ofte vil vi ha behov for tilfeldighet i programmene våre, for eksempel i spill, simuleringer eller ved generering av sikre passord. `random`-biblioteket i Python inneholder flere funksjoner som returnerer tilfeldige tall eller utvalg.

#### **randint(a,b)**

returnerer et tilfeldig heltall mellom **a** og **b** (inklusive begge endepunktene), for eksempel når vi ønsker å simulere terningkast.

```
from random import randint

terningkast = randint(1, 6)
print(f"Terlingkast: {terningkast}")

✓ 0.0s
Terlingkast: 3
```

#### **randrange(start, stop, step)**

returnerer et tilfeldig tall fra sekvensen produsert av `range(start, stop, step)`.

La oss si en kunde kan trekke en lapp med tilfeldig rabatt: 10, 15, 20, 25 eller 30.

```
from random import randrange

tilfeldig_rabatt = randrange(10, 31, 5)
print(f"Terliggig rabatt: {tilfeldig_rabatt}%")

✓ 0.0s
Tilfeldig rabatt: 25%
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### **choice(seq)**

returnerer et tilfeldig element fra en sekvens (f.eks. en liste eller tekst)

```
from random import choice

muligheter = ["øst", "vest", "nord", "sør"]
tilfeldig_retning = choice(muligheter)
print(f"Tilfeldig retning: {tilfeldig_retning}")

✓ 0.0s
Tilfeldig retning: vest
```

### **choices(seq, k=n)**

returnerer en liste med n tilfeldige elementer fra en sekvens (med tilbakelegging).

Sekvens kan være en liste, streng, tuppel, eller annet \*itererbart objekt.

```
from random import choices

frukt = ["eple", "banan", "kiwi", "pære"]
tilfeldig_frukt = choices(frukt, k=2)
print(f"Tilfeldig frukt: {tilfeldig_frukt}")
tilfeldig_frukt = choices(frukt, k=8)
print(f"Tilfeldig frukt: {tilfeldig_frukt}")
# 70 % sannsynlighet for kiwi, 10 % for de andre
tilfeldig_frukt = choices(frukt, weights=[1, 1, 7, 1], k=8)
print(f"Tilfeldig frukt: {tilfeldig_frukt}")

✓ 0.0s
Tilfeldig frukt: ['kiwi', 'banan']
Tilfeldig frukt: ['kiwi', 'eple', 'banan', 'kiwi', 'eple', 'pære', 'eple', 'banan']
Tilfeldig frukt: ['kiwi', 'kiwi', 'kiwi', 'banan', 'kiwi', 'pære', 'kiwi', 'kiwi']
```

### **sample(population, k)**

Returnerer en liste med **k** unike tilfeldige elementer fra en samling (uten tilbakelegging). Gir feil hvis **k** er større enn antall elementer i samlingen.

```
from random import sample

farger = ["rød", "blå", "grønn", "gul", "oransje", "lilla", "brun", "svart", "hvit"]
utvalg = sample(farger, 3)
print(f"Utvalg: {utvalg}")

✓ 0.0s
Utvalg: ['blå', 'oransje', 'grønn']
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### shuffle(x)

Stokker sekvensen **x** slik at elementene får en tilfeldig rekkefølge.

Endrer den opprinnelige listen direkte og returnerer **None**.

```
from random import shuffle

kortstokk = [
    f"{farge} {tall}"
    for farge in ["Hjerter", "Ruter", "Kløver", "Spar"]
    for tall in range(1, 14)
]
print(f"Kortstokk: {kortstokk}")
shuffle(kortstokk)
print(f"Kortstokk: {kortstokk}")

Kortstokk: ['Hjerter 1', 'Hjerter 2', 'Hjerter 3', 'Hjerter 4', 'Hjerter 5',
Kortstokk: ['Spar 3', 'Spar 5', 'Ruter 9', 'Hjerter 4', 'Spar 2', 'Ruter 10',
```

### random()

returner et tilfeldig desimaltall mellom 0 og 1.

Denne funksjonen brukes ofte til sannsynlighetsbeslutninger.

```
from random import random

p = random()
if p < 0.5:
    print("Kron")
else:
    print("Mynt")
Kron
```

### uniform(a,b)

returner et tilfeldig desimaltall, t, mellom a og b, hvor  $a \leq t < b$ .

```
from random import uniform

sann_temperatur = 20
målerfeil = 0.5
simulert_temperatur = uniform(sann_temperatur - målerfeil, sann_temperatur + målerfeil)
print(f"Simulert temperatur: {simulert_temperatur:.1f}")

Simulert temperatur: 19.7
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### seed(x)

hvor **x** er en intitalverdi (frø) for de tilfeldige tallene eller utvalgene. Ved bruk av samme frø får vi identiske sekvenser av "tilfeldige" tall ved hver kjøring, som sikrer reproducerebare resultater. Når vi bruker seed(), setter vi normalt frøet én gang i starten av programmet.

```
from random import seed

for _ in range(3):
    seed(235)
    print(randint(1, 6))
seed() # Tilbakestiller seed for neste kjøring
print(randint(1, 6))

✓ 0.0s

3
3
3
5
```

Vurderingsseksemplar

## 5.3 Sortering

### Innhold

Sorteringsfunksjoner .....	331
Boblesortering .....	332
Utvalgssortering .....	332
Innstikksortering .....	332

### Sorteringsfunksjoner

Vi møtte `sort`-metoden til `list`-klassen i bok 1.5 Lister. Den opprinnelige rekkefølgen går tapt hvis vi ikke lager en kopi av listen. Den innebygde funksjonen `sorted()` returnerer en ny, sortert liste.

```
Når vi sorterer en liste med sort,  
endres rekkefølgen permanent
```

```
tabell = [4,2,5,3,7,1,9,8,6]  
print(tabell)  
tabell.sort()  
print(tabell)
```

✓ 0.0s  
[4, 2, 5, 3, 7, 1, 9, 8, 6]  
[1, 2, 3, 4, 5, 6, 7, 8, 9]

```
Vi kan sortere en kopi av listen  
dersom vi ønsker å beholde den opprinnelige rekkefølgen
```

```
tabell = [4,2,5,3,7,1,9,8,6]  
print(tabell)  
sortert_tabell = tabell.copy()  
sortert_tabell.sort()  
print(sortert_tabell)  
print(tabell)
```

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

```
Da er det enklere å bruke den innebygde  
funksjonen sorted til å lage en ny, sortert liste
```

```
tabell = [4,2,5,3,7,1,9,8,6]  
print(tabell)  
sortert_tabell = sorted(tabell)  
print(sortert_tabell)  
print(tabell)
```

✓ 0.0s  
[4, 2, 5, 3, 7, 1, 9, 8, 6]  
[1, 2, 3, 4, 5, 6, 7, 8, 9]  
[4, 2, 5, 3, 7, 1, 9, 8, 6]

I avsnittet om [anonyme funksjoner](#) lærte vi også å skrive skreddersydde funksjoner for sorteringskriterier.

Vi har også lært å bruke `sort_values`-metoden på et Pandas `DataFrame`-objekt. Denne metoden returnerer også en ny `DataFrame`.

```
import pandas as pd  
navn = ["Per", "Kari", "Ole", "Lise", "Mona", "Nils"]  
alder = [23, 34, 12, 54, 43, 21]  
df = pd.DataFrame({"Navn":navn, "Alder":alder})  
df_sortert = df.sort_values("Alder")  
display(df_sortert)
```

	Navn	Alder
2	Ole	12
5	Nils	21
0	Per	23
1	Kari	34
4	Mona	43
3	Lise	54

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Boblesortering

Men hva om sorteringsmetodene ikke fantes, og vi måtte skrive dem selv? Boblesortering er en enkel sorteringsalgoritme. Vi sammenligner og bytter om på to og to elementer i listen etter behov. Etter én runde er det siste elementet på riktig plass. I neste runde gjør vi det samme, men stopper ved nest siste element i listen, slik at det nå er to elementer på riktig plass. Slik fortsetter det. Her er det unødvendig å kjøre de fire siste rundene, så denne algoritmen kan forbedres.

### Utvalgssortering

Utvalgssortering møtte vi i løsningsforslaget til [øvingsoppgave 2.7.3](#). Først finner vi indeksen til elementet med minst verdi. Deretter bytter vi det første elementet med elementet med minst verdi. Videre fortsetter vi fra det andre elementet og finner indeksen til det minste elementet blant de gjenværende elementene. Deretter bytter vi elementet i posisjon to med dette elementet. Vi gjentar denne prosessen for hvert gjenværende element i listen. 1. runde gjentas n ganger. 2. runde gjentas n-1 ganger. 3. runde gjentas n-2 ganger, og den siste (n-te) runden gjennomføres kun én gang. Vi legger merke til at noen av de siste rundene kjøres unødvendig, noe som gir rom for forbedring av algoritmen. Dette kan løses ved å innføre et "flagg" som indikerer om det har skjedd noen bytter i runden. Hvis det ikke har det, er vi ferdige. Her bytter først 1- og 4-tallet plass, så skjer det ingen ting, for 2-tallet står på riktig plass. Deretter bytter 3- og 5-tallet plass, 4 og 5, osv.

### Innstikksortering

Innstikksortering er en algoritme som minner om å sortere kort. Denne metoden sorterer elementene i en liste ved å plassere hvert element på riktig plass blant de tidligere sorterte elementene. Vi begynner med det andre elementet i listen og sammenligner det med det første elementet. Hvis det andre elementet er mindre enn det første, bytter vi plass på dem. Deretter tar vi det tredje elementet og setter det på riktig plass blant de to første elementene vi allerede har sortert. Vi fortsetter på samme måte med de påfølgende elementene i listen. Vårt eksempel begynner med at 4 og 2 bytter plass, slik at den nye tabellen blir [2, 4, 5, 3, 7, 1, 9, 8, 6]. Deretter er 5 allerede på riktig plass, så ingenting skjer i den andre runden. I den tredje runden skal 3 settes inn mellom 2 og 4, slik at 4 og 5

```
Boblesortering

tabell = [4, 2, 5, 3, 7, 1, 9, 8, 6]
print(tabell)
n = len(tabell)
for i in range (n):
    for j in range (0,n-i-1):
        if tabell[j]>tabell[j+1]:
            tabell[j], tabell[j+1] = tabell[j+1], tabell[j]
    print(tabell)

✓ 0.1s

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

```
utvalgssortering

tabell = [4, 2, 5, 3, 7, 1, 9, 8, 6]
print(tabell)
n = len(tabell)
for i in range (n-1):
    minste = i
    for j in range (i+1,n):
        if tabell[minste]>tabell[j]:
            minste = j
    tabell[i], tabell[minste] = tabell[minste], tabell[i]
print(tabell)

✓ 0.0s

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

```
Innstikksortering

tabell = [4, 2, 5, 3, 7, 1, 9, 8, 6]
print(tabell)
n = len(tabell)
for i in range (1,n):
    verdi = tabell[i]
    j = i-1
    while j>=0 and verdi<tabell[j]:
        tabell[j+1] = tabell[j]
        j -= 1
    tabell[j+1] = verdi
print(tabell)

✓ 0.0s

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

## **Smidig IT-2**

### **for eksklusiv bruk ved <navn på skole>**

forskyves ett hakk mot høyre. Den nye tabellen blir da [2, 3, 4, 5, 7, 1, 9, 8, 6].

Ingen av disse sorteringsalgoritmene er spesielt effektive. Boblesortering, utvalgssortering og innstikksortering er enkle og intuitive sorteringsmetoder, men de kan bli svært tidkrevende for store lister. Selv om det finnes mer avanserte sorteringsalgoritmer som håndterer større datasett mer effektivt, får det ligge til senere studier.

Vurderingsseksemplar

## 5.4 Rekursjon

### Innhold

Rekursjon forklart med Fibonaccitallene .....	334
Tårnene i Hanoi .....	334
Tidsmålinger .....	335

### Rekursjon forklart med Fibonaccitallene

I [oppgave 1.4.5](#) om løkker skulle vi skrive ut alle Fibonaccitallene under 1000. I eksempelet under til venstre bruker vi en løkke til å beregne det n-te Fibonaccitallet. Imidlertid finnes det en annen tilnærming for å løse slike problemer. Vi kan bruke rekursjon som vil si at en funksjon kaller seg selv. Det er viktig å bruke betingelser som hindrer uendelige kall, som kan føre til at programmet "henger" eller krasjer. Rekursjon er en kraftig metodikk som er spesielt nyttig når et problem kan brytes ned i mindre, lignende delproblemer, slik som i dette tilfellet med Fibonaccitallene. Programmet blir lite og lett å forstå, men det krever mer minne og tar lengre tid å kjøre. Når en funksjon kaller seg selv i en rekursiv kjede, starter ikke beregningen før den innerste funksjonen returnerer et svar til den neste funksjonen i kjeden, som deretter returnerer til den neste funksjonen osv. I koden under til høyre bruker vi rekursjon.

Fibonacci tall med for-løkke

```
def fibonacci_iterativ(n):
    if n <= 1:
        return n
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

for i in range(15):
    print(fibonacci_iterativ(i), end=" ")
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

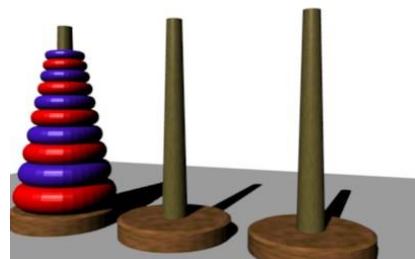
Fibonacci med rekursjon

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

for i in range(15):
    print(fibonacci_rekursiv(i), end=" ")
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

### Tårnene i Hanoi

Et annet klassisk problem som løses elegant med rekursjon, er [Tårnene i Hanoi](#). Dette puslespillet består av tre pinner og en samling runde skiver med et hull i midten. Skivene varierer i bredde og kan plasseres på en av de tre pinnene. Når spillet starter, er alle skivene ordnet på en pinne i konisk form, med den minste skiven øverst. Målet med puslespillet er å flytte alle skivene til en annen pinne ved å følge noen enkle regler:



## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

- Vi kan kun flytte én skive av gangen.
- Den øverste skiven på en pinne kan flyttes til toppen av en annen pinne, forutsatt at det ikke er noen mindre skiver som forhindrer dette.
- Det er ikke tillatt å plassere en skive oppå en mindre skive.

Delproblemet er å flytte alle bortsett fra den nederste skiven ( $n-1$ ) til en hjelpestolpe. Deretter kan vi flytte den nederste skiven til destinasjonsstolpen. Så gjentar vi delproblemet og flytter de  $(n-1)$  skivene fra hjelpestolpen oppå den  $n$ -te skiven på destinasjonsstolpen.

```
Tårnene i Hanoi

def tårnene_i_hanoi(n, kilde, destinasjon, hjelp):
    if n == 1:
        print("Flytt skive 1 fra", kilde, "til", destinasjon)
        return
    tårnene_i_hanoi(n - 1, kilde, hjelp, destinasjon)
    print("Flytt skive", n, "fra", kilde, "til", destinasjon)
    tårnene_i_hanoi(n - 1, hjelp, destinasjon, kilde)

n = 3
# A, B, C er navnene på stolpene
tårnene_i_hanoi(n, 'A', 'B', 'C')

Flytt skive 1 fra A til B
Flytt skive 2 fra A til C
Flytt skive 1 fra B til C
Flytt skive 3 fra A til B
Flytt skive 1 fra C til A
Flytt skive 2 fra C til B
Flytt skive 1 fra A til B
```

### Tidsmålinger

Selv om rekursjon er intuitivt og elegant, krever det mer minne og tar lengre tid å kjøre enn en iterativ løsning, som eksempelvis med Fibonaccitallene og Tårnene i Hanoi, hvor funksjonen kaller seg selv to ganger. Dermed øker tiden eksponentielt fordi antall kall dobles seg for hvert nivå i rekursjonen: 1, 2, 4, 8, 16, osv. For Fibonaccitallene har vi målt tiden det tar å kjøre rekursivt og iterativt 1000 ganger. Den iterative løsningen brukte ca. 1 ms, mens rekursjonen tok omtrent 1 sekund. Rekursjonen var over 1000 ganger tregere enn den iterative metoden. Vi må altså vurdere hvorvidt vi har tilstrekkelige ressurser og hvor lenge vi er villige til å vente på resultatet for å kunne benytte en rekursiv løsning, selv om den fremstår enkel.

```
import timeit

# Måling av iterativ funksjon
iterativ_tid = timeit.timeit('fibonacci_iterativ(20)', globals=globals(), number=1000)

# Måling av rekursiv funksjon
rekursiv_tid = timeit.timeit('fibonacci_rekursiv(20)', globals=globals(), number=1000)

print(f"Tid for iterativ tilnærming: {iterativ_tid} sekunder")
print(f"Tid for rekursiv tilnærming: {rekursiv_tid} sekunder")

✓ 1.0s
Tid for iterativ tilnærming: 0.001049500000590342 sekunder
Tid for rekursiv tilnærming: 1.0117874999996275 sekunder
```

Prøv å gjøre en tilsvarende sammenligning for fakultetberegninger, hvor den rekursive funksjonen kaller seg selv bare én gang.

## 5.5 Avansert JSON

### Innhold

Innledning .....	336
Dypere hierarki .....	336
Flere dimensjoner.....	338

### Innledning

Da vi introduserte JSON, lærte vi at de grunnleggende byggeklossene i JSON var objekter (navn/verdi-par) og ordnede lister. Disse tilsvarer henholdsvis **dictionary** og **list** i Python. Vi lærte også om andre datatyper som string, number, true og false som konverteres til henholdsvis **str**, **int** eller **float**, **True** og **False** i Python. JSONs hierarkiske struktur gir mulighet for å inneholde objekter og lister inni andre objekter og lister, og dette danner grunnlaget for mer komplekse oppsett. Vi vil nå utforske JSONs hierarkiske oppbygging og se hvordan vi kan skape enda mer sofistikerte strukturer. Vi vil se på hvordan dypere hierarkier dannes og behandles, samt hvordan vi kan organisere data etter flere dimensjoner

### Dypere hierarki

Vi skal utforske dypere hierarkier i JSON med eksempelet i filen **person.json**, vist til høyre på skjermbildet.. Dette eksempelet presenterer en klar hierarkisk struktur. På det øverste nivået finner vi en nøkkel kalt **person**, som igjen inneholder en ordbok med flere undernøkler. Både **navn** og **alder** har verdier av enkle datatyper, mens **adresse** er en undernøkkel med enda en ordbok og dens egne undernøkler. Denne organiseringen av data reflekterer en hierarkisk tilnærming, hvor hver nøkkel kan inneholde enten en enkelt verdi eller en annen understruktur. Vi skal bruke denne strukturen for å lage en pandas **DataFrame** og utforske hvordan vi kan få tilgang til de innebygde verdiene.

Etter at vi hadde lært om pandas, fant vi det enklest å bruke **pandas.read\_json()** til å lese inn data fra filer i JSON format. Når det er flere nivåer i en hierarkisk struktur, oppstår det utfordringer med å representere dataene på en flat måte. Her får vi en tabell med kolonnen **person** og radindeksene **adresse**, **alder** og **navn** hvor adressen er en ordbok. Det fører til kompleks indeksering og gjør det vanskeligere å få tilgang til verdier i de innebygde objektene. Se filen **t\_5\_5\_1\_person.ipynb**.

Her kan imidlertid valget av **json.load()** være mer fordelaktig. Hvis vi vil se alle nøkkel-verdi-parene, kan vi bruke en løkke med rekursive kall til å iterere gjennom dataene. Dette er spesielt nyttig hvis vi ikke kjenner den nøyaktige strukturen på forhånd. En slik funksjon og utskriften er vist i skjermbildene på neste side.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Rekursiv funksjon for å skrive ut alle nøkkel-verdi-parene i JSON data:

```
# Utforsk JSON-strukturen med nøkkel-verdi-par og rekursjon
import json
from pathlib import Path

sti = Path(__file__).resolve().parent
fil = sti.joinpath('person.json')

# Les inn JSON-filen
with open(fil) as f:
    json_data = json.load(f)

print(json_data)

def utforsk_json(data, level=0):
    if isinstance(data, dict):
        for key, value in data.items():
            innrykk = " " * level
            if isinstance(value, dict):
                # Hvis verdien er nok en ordbok, utforsk den rekursivt
                print(f"{innrykk}{key}:")
                utforsk_json(value, level + 1)
            else:
                # Hvis ikke, skriv ut nøkkel og verdi på samme linje
                print(f"{innrykk}{key}: {value}")
    else:
        # Hvis det ikke er en ordbok, skriv ut verdien direkte
        print(data)

utforsk_json(json_data)

{'person': {'navn': 'Ola Nordmann', 'alder': 30, 'adresse': {'gate': 'Storgata 10', 'by': 'Mysen', 'postnummer': '1850'}}}
person:
navn: Ola Nordmann
alder: 30
adresse:
  gate: Storgata 10
  by: Mysen
  postnummer: 1850
```

`json.load()` gir oss friheten til å skreddersy strukturen nøyaktig etter våre behov. Dette kommer særlig godt med når vi ønsker å tilpasse dataene til den aktuelle oppgaven vi arbeider med, eller når vi har klart definerte mål som vi ønsker å oppnå. I programmet vist i skjermbildet til høyre oppretter vi en **DataFrame** med kolonnene **Navn**, **Alder**, **Gate**, **By** og **Postnummer**.

```
import pandas as pd
import json

# Les inn JSON-filen
with open('person.json') as f:
    person_data = json.load(f)

# Hent ut dataene
navn = person_data['person']['navn']
alder = person_data['person'][['alder']]
gate = person_data['person'][['adresse']]['gate']
by = person_data['person'][['adresse']]['by']
postnummer = person_data['person'][['adresse']]['postnummer']

# Lag en Pandas DataFrame
data = {'Navn': [navn], 'Alder': [alder], 'Gate': [gate],
        'By': [by], 'Postnummer': [postnummer]}
df = pd.DataFrame(data)

# Vis innholdet i tabellen
display(df)
✓ 0.0s
```

	Navn	Alder	Gate	By	Postnummer
0	Ola Nordmann	30	Storgata 10	Mysen	1850

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Flerere dimensjoner

Da vi møtte på komplekse, flerdimensjonale JSON data i tidligere tilfeller, som i JSON-stat formatet, brukte vi `pyjstat` biblioteket til å hjelpe oss. Vi skal nå ta for oss et litt enklere eksempel for å se hvordan vi danner og behandler slike strukturer. Vi benytter filen `salg.json` som gir oss informasjon om salg fordelt på ulike regioner og år. Nivå 1 er `datasett` som inkluderer nivå 2 med `dimesjon` og `verdi`. Inne i `dimensjon` finner vi nivå 3 med `region` og `år`, hver med sine underkomponenter `beskrivelse` og `kategori` på nivå 4. Denne hierarkiske strukturen med flere nivåer av informasjon, inkludert bruk av lister som holder verdier som regioner og år, er karakteristisk for flerdimensjonale data.

```
↳ salg.json > ...
1  {
2      "datasett": {
3          "dimensjon": {
4              "region": {
5                  "beskrivelse": "Region",
6                  "kategori": ["Nord", "Sør", "Øst", "Vest"]
7              },
8              "år": {
9                  "beskrivelse": "År",
10                 "kategori": ["2019", "2020", "2021"]
11             }
12         },
13         "verdi": [100, 150, 200, 250, 300, 350, 400, 450, 500]
14     }
15 }
```

```
import pandas as pd
import json

# Les inn JSON-data
with open('salg.json', encoding='utf-8') as json_file:
    json_data = json.load(json_file)

# Hent kategorier og verdier
regioner = json_data['datasett']['dimensjon']['region']['kategori']
år = json_data['datasett']['dimensjon']['år']['kategori']
verdier = json_data['datasett']['verdi']

# Lag en Pandas DataFrame
data = [(regioner[i // len(år)], år[i % len(år)], verdi) \
        for i, verdi in enumerate(verdier)]
df = pd.DataFrame(data, columns=['Region', 'År', 'Salg'])

# Vis tabellen
display(df)
```

	Region	År	Salg
0	Nord	2019	100
1	Nord	2020	150
2	Nord	2021	200
3	Sør	2019	250
4	Sør	2020	300
5	Sør	2021	350
6	Øst	2019	400
7	Øst	2020	450
8	Øst	2021	500

Etter å ha lest inn filen henter vi først ut listene `regioner`, `år` og `verdier`. Deretter bygger vi opp en ny liste, `data`, med tupler som inneholder verdiene for `region`, `år` og `verdi`. Denne data listen brukes til å lage en `DataFrame` med kolonnene `Region`, `År` og `Salg`. Se filen `t_5_5_2_salg.ipynb`.

Vurderingsseksempler

## 5.6 Etterligning (mocking) med pytest

### Innhold

Introduksjon .....	339
Etterligning av tilfeldige verdier.....	339
Etterligning av nedlasting fra Internett .....	340

### Introduksjon

Når vi enhetstester, tester vi én enhet om gangen, enten det er en funksjon, metode, klasse eller modul. Noen ganger er denne enheten avhengig av annen kode som vi ikke kan kontrollere, for eksempel en funksjon som returnerer tilfeldige verdier, aksjekurser i sanntid, nettverkstilgang eller lignende. I slike tilfeller kan vi bruke *mocking* for å erstatte den ukontrollerbare koden med en simulert versjon som gir forutsigbare resultater. I kapittel [4.8 Mer om testing](#) brukte vi `mocker-fixturen` i pytest til å teste en klasse uavhengig av objekter og metoder i en annen klasse. Vi kan også bruke `mocker` til å etterligne tilfeldige verdier og nedlasting fra Internett. Husk at for å bruke `mocker` i pytest må `pytest-mock` være installert med `pip install pytest-mock`.

### Etterligning av tilfeldige verdier

I det første eksempelet har vi en funksjon, `tilfeldig_farge`, som returnerer 'Rød', 'Grønn' eller 'Blå' basert på et tilfeldig valg fra en liste med farger ved hjelp av `random.choice()`.

Når vi skal teste `tilfeldig_farge()`, bytter vi ut `random.choice()` med en etterligning som returnerer forutsigbare verdier. Dette gjør vi ved hjelp av `mocker.patch()`. Hvis vi hadde brukt `from random import choice` i programmet som skal testes, måtte vi ha ventet med importen til etter at *fixturen* var satt opp. Hvis ikke, ville `random.choice()` kjørt som vanlig og returnert tilfeldige verdier, noe som hadde gjort testen upålіtelig.

Ved å bruke `return_value` på `choice_mock`, kan vi kontrollere hvilken farge `tilfeldig_farge()` skal returnere. Dette gjør testen deterministisk og forutsigbar. Vi gjør ingen endringer i programmodulen som fremdeles returnerer vilkårlige farger når den brukes for seg selv eller importeres av andre. Det er kun testprogrammet som bytter ut `random.choice()` med noe forutsigbart når testen kjøres. Se mappen [5\\_06\\_Etterligning\\_med\\_pytest](#).

```
import random

def tilfeldig_farge() -> str:
    """Returner en vilkårlig farge."""
    farger = ["Rød", "Grønn", "Blå"]
    return random.choice(farger)

if __name__ == "__main__":
    print(tilfeldig_farge())
```

```
import tilfeldig
import pytest

@pytest.fixture
def choice_mock(mocker):
    """Fixture for å mоке random.choice med fort"""
    return mocker.patch("random.choice")

def test_tilfeldig_farge(choice_mock):
    choice_mock.return_value = "Rød"
    assert tilfeldig.tilfeldig_farge() == "Rød"

    choice_mock.return_value = "Grønn"
    assert tilfeldig.tilfeldig_farge() == "Grønn"

    choice_mock.return_value = "Blå"
    assert tilfeldig.tilfeldig_farge() == "Blå"

if __name__ == "__main__":
    pytest.main(["-v", "-s"])
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Etterligning av nedlasting fra Internett

I det andre eksempelet laster vi ned en nettside som vi analyserer med [Beautiful Soup](#). Beautiful Soup er et bibliotek som gjør det enkelt å skrape informasjon fra nettsider. Se [dokumentasjonen](#). Installer biblioteket med  
`pip install beautifulsoup4`

```
import requests
from bs4 import BeautifulSoup

def hent_tittel(url: str) -> str:
    """Last ned en valgfri webside og returner tittelen."""
    try:
        response = requests.get(url)
        if response.status_code != 200:
            return response.reason
        soup = BeautifulSoup(response.text, "html.parser")
        return soup.title.string
    except requests.exceptions.RequestException as e:
        raise SystemExit(e) from e

if __name__ == "__main__":
    url = "https://en.wikipedia.org/wiki/Unit_testing"
    print(hent_tittel(url))
```

Her henter vi ut bare tittelen. Det er enkelt å teste programmet, men hva hvis vi ikke har en nettverksforbindelse? Da feiler testen. Må vi utsette testingen? Ikke hvis vi har lagret websiden og bruker den lagrede kopien i stedet.

Vi skriver en *fixture* `mock_response` som etterligner `requests.get()` i `hent_tittel()`. Denne leser inn den lagrede websiden som tekst og oppretter et `MagicMock`-objekt som etterligner `requests.Response`. Deretter settes objektets `status_code` til 200 og `text`-attributtet settes til teksten vi leste inn. Vi bruker `mocker.patch` til å erstatte returverdien fra `requests.get` med vår `mock_response`. Testen passerer nå uten nettverksforbindelse, og `hent_tittel()` fungerer fremdeles for vilkårlige websider.

```
import nedlasting
import requests
import pytest
from pathlib import Path

@pytest.fixture
def mock_response(mocker):
    """
    Fixture for å etterligne requests.get responsen med innhold fra en lokal HTML-fil.
    Dette sikrer at testen ikke avhenger av en faktisk nettverksforbindelse.
    """
    fil = Path(__file__).parent.resolve().joinpath("Unit_testing.htm")
    with open(fil, encoding="utf-8") as f:
        mock_html_content = f.read()

    # MagicMock brukes til å lage et mock-objekt som etterligner requests.Response
    mock_response = mocker.MagicMock(spec=requests.Response)
    mock_response.status_code = 200
    mock_response.text = mock_html_content
    mock_response.raise_for_status = mocker.MagicMock()

    # mocker.patch erstatter returverdien fra requests.get med vår mock_response
    mocker.patch("requests.get", return_value=mock_response)

def test_hent_tittel(mock_response):
    """
    Selv om mock_response ikke brukes direkte i denne funksjonen,
    er argumentet nødvendig for å sikre at requests.get blir mocket korrekt
    av pytest-fixturen. Dette gjør at testen kan kjøre uavhengig av
    nettverksforbindelse.
    """
    url = "https://en.wikipedia.org/wiki/Unit_testing"
    expected_title = "Unit testing - Wikipedia"
    tittel = nedlasting.hent_tittel(url)
    assert tittel == expected_title

if __name__ == "__main__":
    pytest.main(["-v", "-s"])
```

```
test_nedlasting.py::test_hent_tittel PASSED
test_tilfeldig.py::test_tilfeldig_farge PASSED

===== 2 passed in 0.09s =====
```

Vurderingsseksempler

## 5.7 Unittest

### Innhold

Unittest og bruk av <i>Test Fixtures</i> .....	341
Kjøring av testene.....	342

### Unittest og bruk av *Test Fixtures*

I avsnittet Testkontekst i [3.7 Enhetstesting av objektorienteerte programmer](#) utførte vi tester med `pytest` på en `Bil`-klasse, hvor vi brukte *fixtures* for å sette opp og rydde opp etter testene. Her viser vi hvordan de samme testene kan utføres med *unittest*, som har sitt eget system for å opprette testtilfeller og *test fixtures*.

I unittest må testtilfellene opprettes i en klasse som arver fra superklassen `unittest.TestCase`.

- `TestCase` er superklassen til et testtilfelle
- `TestSuite` samler testtilfeller som skal kjøres sammen.
- `TextTestRunner` Kjører testene og gir tilbakemelding om resultatene.
- *Test Fixture* er forberedelser som er nødvendige for å gjennomføre testen(e). Det kan være å opprette midlertidige mapper, databaser eller start en server. Etter testene er ferdige, må det ryddes opp.
  - `setUpClass` er en klassemetode som kjøres én gang før testene i klassen.
  - `setUp` er en metode som kjøres før hver testmetode i klassen.
  - `tearDown` er en metode som kjøres etter hver testmetode i klassen.
  - `tearDownClass` er en klassemetode som kjøres én gang etter testene i klassen er ferdige

```
# test_bil.py
import unittest
from bil import Bil

class TestBil(unittest.TestCase):
    @classmethod
    def setUpClass(cls) -> None:
        print("setUpClass")
        return super().setUpClass()

    @classmethod
    def tearDownClass(cls) -> None:
        print("\ntearDownClass")
        return super().tearDownClass()

    def setUp(self) -> None:
        print("\nsetUp")
        self.bil = Bil()
        return super().setUp()

    def tearDown(self) -> None:
        print("tearDown")
        return super().tearDown()

    def test_start_motor(self):
        print("Tester start av motor")
        self.bil.start_motor()
        assert self.bil.motor_status == "startet"

    def test_stopp_motor(self):
        print("Tester stopp av motor")
        self.bil.start_motor()
        self.bil.stopp_motor()
        assert self.bil.motor_status == "stoppet"

if __name__ == "__main__":
    suite = unittest.TestSuite()
    suite.addTest(TestBil("test_start_motor"))
    suite.addTest(TestBil("test_stopp_motor"))
    runner = unittest.TextTestRunner()
    runner.run(suite)
```

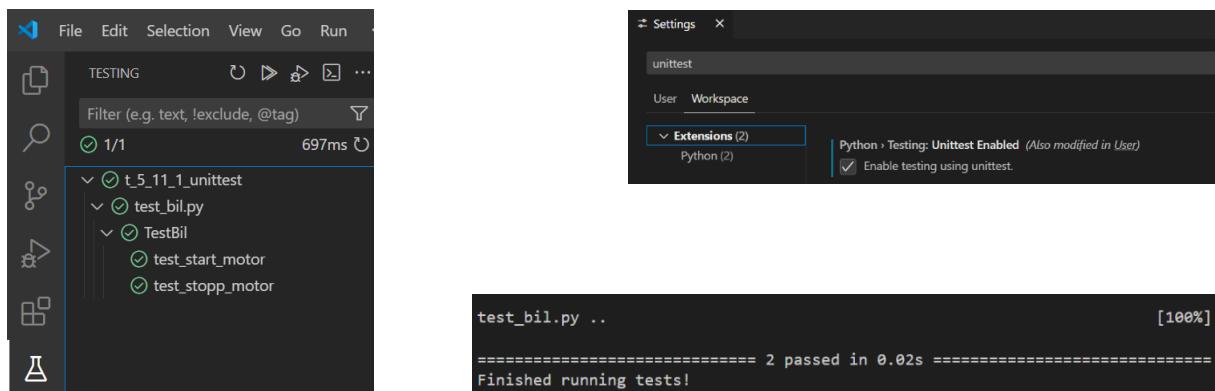
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi har med noen av disse fire metodene, kjøres de automatisk. I eksempelet i mappen `t_5_07_unittest` har vi tatt med alle sammen, men vi bruker bare `setUp` for å slippe å opprette bilen i hver test. De andre metodene skriver bare ut deres egne navn, for å vise at de blir kalt.

### Kjøring av testene

For å sette opp og kjøre testene i VS Code må alternativet *Enable testing using unittest* være aktivert i *Settings* (Ctrl+,)



Alternativt kan vis skrive `python -m unittest` i terminalvinduet for å se hele utskriften og ikke bare testresultatene.

Det får vi også til ved å kjøre testmodulen (ved å klikke på pilen *Run Python File*) med følgende kode under `if __name__ == "__main__":`:

```
if __name__ == "__main__":
    suite = unittest.TestSuite()
    suite.addTest(TestBil('test_start_motor'))
    suite.addTest(TestBil('test_stopp_motor'))
    runner = unittest.TextTestRunner()
    runner.run(suite)
```

```
setUpClass
test_fullt_navn (__main__.TestElev) ... setUp
test_fullt_navn
tearDown
ok
test_hilsen (__main__.TestElev) ... setUp
test_hilsen
tearDown
ok
tearDownClass

-----
Ran 2 tests in 0.002s
OK
```

Eller bare:

```
if __name__ == "__main__":
    unittest.main()
```

Vurderingsseksempler

## 5.8 Designmønste

### Innhold

Introduksjon .....	343
Singleton.....	344
Factory Method.....	344
Decorator.....	345
MVC .....	346

### Introduksjon

Forskjellige systemer som løser ulike oppgaver, har naturlig nok ulik kode, men også flere fellestrekks, som hvordan koden er bygget opp og organisert. [Designmønstre](#) er veiledninger for hvordan vi kan utforme koden for å løse problemer som ofte oppstår i objektorientert programmering. De løser mer generelle problemer enn hjelpen vi får fra detaljerte algoritmer eller spesifikke biblioteker. Ved å følge disse velprøvde mønstrene, blir koden vår lettere å lese, vedlikeholde, utvide og gjenbruke. Husk at design handler om "hvordan", ikke om "hva" som i analyse.

Å utvikle objektorienterte programmer er vanskelig, fordi det er utfordrende å finne ut hva som skal bli til objekter, attributter, metoder og forholdene mellom klassene. Å lage systemer som både er enkle å vedlikeholde, videreutvikle og gjenbruke, er enda vanskeligere. Det er her designmønstre kommer inn. De er ikke for nybegynnere, for grunnleggende forståelse av objektorientert programmering må være på plass, som objekter, metoder, attributter, klasser, arv og andre relasjoner mellom klasser, innkapsling, polymorfisme, osv. Når erfarne utviklere har funnet en metode som virker, bruker de den om igjen. Den blir til et mønster. Se på det som maler som kan brukes på hele eller deler av programmet.

Designmønstre ble populært i objektorientert programmering etter at boka *Design Patterns: Elements of Reusable Object-Oriented Software* ble utgitt i 1994 av den såkalte *Gang of Four* (*GoF*): Erich Gamma, Richard Helm, Ralph Johnson og John Vlissides. Boka introduserte 23 designmønstre som gir standardiserte løsninger på gjentakende problemer i programvareutvikling. De gjør det enklere å håndtere vanskelige problemer ved å bruke gjennomprøvde løsninger som har blitt testet over tid, fremmer gjenbruk av løsninger og kode, som kan redusere utviklingstiden, og gjør det lettere for utviklere i et prosjekt å forstå hverandre og kommunisere effektivt ved å bruke en felles terminologi.

Her har vi valgt ut fire kjente designmønstre for å vise hvordan slike mønstre kan se ut, anvendes og implementeres i Python.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Singleton

*Singleton* er et designmønster som sikrer at en klasse kun har én instans. Dersom vi prøver å opprette flere, returneres den første og eneste instansen av klassen. Instanser gir flere fordeler enn statiske klasser med bare klassemетодer, fordi de blant annet kan brukes med arv og polymorfisme og er lettere å teste.

Tenk deg for eksempel at du har en konfigurasjonsfil. Det er viktig at den leses bare én gang, slik at applikasjonen alltid bruker det samme settet med verdier.

Python har en spesiell metode `__new__` som kalles før `__init__` og oppretter en instans av klassen. Denne kan vi overskrive for å implementere *Singleton*-mønsteret. Dersom instansen, som også tas vare på i klasseattributtet `_instans`, eksisterer, returneres denne. Hvis ikke, kalles `__new__`-metoden fra superklassen for å opprette en ny instans. Se [t\\_5\\_8\\_1\\_singleton.py](#)

```
class Singleton:
    _instans = None

    def __new__(cls, *args, **kwargs):
        if cls._instans is None:
            cls._instans = super().__new__(cls)
        return cls._instans

    def __init__(self, konfig):
        self.konfig = konfig

# Eksempel på bruk
if __name__ == '__main__':
    instans1 = Singleton("Verdi1")
    instans2 = Singleton("Verdi2")
    print(f"instans1.konfig: {instans1.konfig}")           # Verdi2
    print(f"instans2.konfig: {instans2.konfig}")           # Verdi2
    print(f"instans1 == instans2: {instans1 == instans2}") # Samme verdier?
    print(f"id(instans1): {id(instans1)}")
    print(f"id(instans2): {id(instans2)}")
    print(f"instans1 is instans2: {instans1 is instans2}") # Samme objekt?
    instans1.konfig = "NyVerdi"
    print(f"instans2.konfig etter endring: {instans2.konfig}") # NyVerdi
```

```
instans1.konfig: Verdi2
instans2.konfig: Verdi2
instans1 == instans2: True
id(instans1): 2204224063776
id(instans2): 2204224063776
instans1 is instans2: True
instans2.konfig etter endring: NyVerdi
```

Her ser vi at verdien til `konfig`-attributtet i `instans1` endres når vi forsøker å opprette `instans2`. Dette er fordi når `__init__()` kalles den andre gangen, får den ikke en ny instans, men instansen som `instans1` peker på. En mer robust versjon finnes i [t\\_5\\_8\\_1\\_singleton\\_robust.py](#). Der brukes en metaklasse med `__call__`-metoden som igjen kaller `__new__()` og `__init__()`. Dette sørger for at kun én instans opprettes og at `__init__()` kjøres bare én gang. Dessuten brukes en ordbok til instanser for å støtte polymorfisme og arv.

### Factory Method

Vi opprettet instanser fra informasjon i en fil ved bruk av en [klassemетод](#) i bolken [2.6 Objektorientert programmering](#). Koden kan gjøres lettere å vedlikeholde og videreutvikle ved å skille ut ansvaret for å opprette instansene til en annen klasse, slik at `Dyr`-klassen kun har ansvaret for å representere dyrene. *Factory Method*-mønsteret definerer et grensesnitt<sup>34</sup> for å opprette et objekt, men lar underklassene bestemme hvilken klasse som skal opprettes. Når underklassene tar ansvar for å opprette objektene, blir det enklere å legge til nye typer

<sup>34</sup> Tenk på et grensesnitt som en avtale om hvilke metoder som må implementeres. Python har ikke grensesnitt, men vi kan bruke abstrakte klasser for å oppnå det samme.

# Smidig IT-2

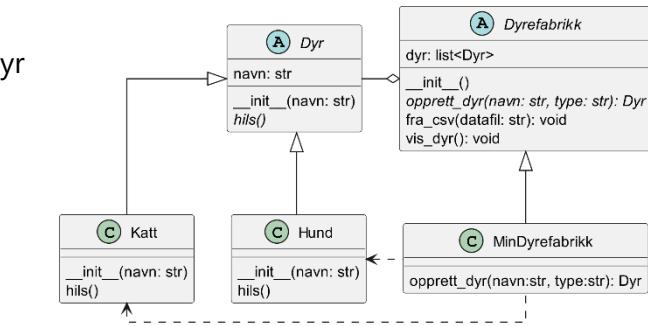
## for eksklusiv bruk ved <navn på skole>

objekter uten å endre eksisterende kode. I vårt eksempel har vi to abstrakte klasser, Dyr og Dyrefabrikk, samt flere konkrete dyreklasser og en konkret MinDyrefabrikk-kasse.

Hvis vi nå legger til klassen Sau i filen t\_5\_8\_2\_factory.py og Sigurd, Sau i filen t\_5\_8\_2\_factory.csv, fungerer programmet uten flere endringer

```
class Sau(Dyr):
    def __init__(self, navn):
        super().__init__(navn)

    def hils(self) -> None:
        print('Bæ! ', end='')
        print(f'Jeg heter {self.navn} og er en sau')
```



```
t_5_8_2_factory.csv > data
1 Hans,Hund
2 Hege,Hund
3 Kristin,Katt
4 Kåre,Katt
5 Knut,Katt
6 Sigurd,Sau
```

```
Voff! Jeg heter Hans og er en hund
Voff! Jeg heter Hege og er en hund
Mjau! Jeg heter Kristin og er en katt
Mjau! Jeg heter Kåre og er en katt
Mjau! Jeg heter Knut og er en katt
Bæ! Jeg heter Sigurd og er en sau
```

I en tradisjonell implementering av *Factory Method*-mønsteret ville hver underklasse av Dyr ha sin egen tilhørende MinDyrefabrikk. I vårt eksempel har vi imidlertid valgt å samle opprettelsen av instanser fra alle dyreklassene i en enkelt MinDyrefabrikk for å forenkle koden. Dette er mulig fordi vi kan få tak i en klasse hvis den er definert, ved å bruke `globals()[type]`, hvor `type` er en tekststreng med navnet på klassen. På denne måten kan vi bruke denne dyreklassen til å opprette en instans av riktig klasse uten å lage separate fabrikker for hver dyreart.

## Decorator

Ønsker vi å utvide egenskapene til enkelte objekter uten å endre hele klassen, kan vi bruke *Decorator*-mønsteret. Vi kunne brukt arv og en underklasse, men det blir en statisk løsning. Med *Decorator* kan vi legge til og fjerne nye egenskaper og metoder mens programmet kjører.

Tenk at vi har en `Melding`-klasse uten formatering av tekst, men at det er ønskelig for noen meldinger. Da kan vi pakke denne klassen inn i en `Dekoratør`-klasse, som vist nedenfor. Men dette er en lite fleksibel løsning, fordi det krever at vi lager en ny klasse for hver kombinasjon av egenskaper vi ønsker å legge til. Se t\_5\_8\_3\_dekorator\_enkel.py

```
Normal melding: Dette er en enkel melding.
Fet melding: Dette er en enkel melding.
Kursiv melding: Dette er en enkel melding.
```

Det er bedre å programmere til et grensesnitt, som implementeres med abstrakte klasser i Python, for i tillegg til fordelene med arv, får vi flere muligheter for polymorfisme. *Decorator*-mønsteret lar oss legge til eller fjerne egenskaper ved å kombinere ulike dekoratørklasser mens programmet kjører. Klassen `Melding` trenger ikke endres for å legge til nye egenskaper og følger dermed

```
class Melding:
    # Utelater andre attributter og metoder for enkelhetens skyld
    def __init__(self, tekst):
        self.tekst = tekst

    def hent_tekst(self):
        return self.tekst

class Dekorator:
    def __init__(self, melding):
        self.melding = melding

    def hent_tekst(self, type="normal"):
        # ANSI-koder for utskrift til terminal
        if type == "fet":
            return f"\033[1m{self.melding.hent_tekst()}\033[0m"
        elif type == "kursiv":
            return f"\033[3m{self.melding.hent_tekst()}\033[0m"
        else:
            return self.melding.hent_tekst()

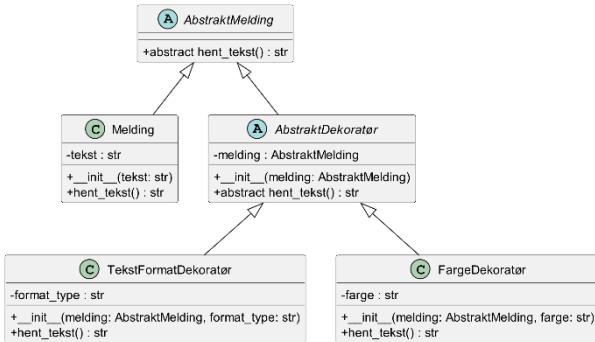
if __name__ == "__main__":
    melding = Melding("Dette er en enkel melding.")
    print("Normal melding: ", melding.hent_tekst())
    print("Fet melding: {Dekorator(melding).hent_tekst('fet')}")
    print("Kursiv melding: {Dekorator(melding).hent_tekst('kursiv')}"
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

prinsippet om å være åpen for utvidelse, men lukket for endring. *Decorator*-mønsteret tillater gjenbruk og reduserer duplisert kode ved å kombinere eksisterende og nye dekoratører for å utvide egenskapene til den opprinnelige klassen. Hver dekoratør har et enkelt ansvar, noe som gjør koden lettere å lese, mer forståelig, enklere å teste og vedlikeholde, reduserer risikoen for feil, forbedrer inndelingen og øker gjenbruk. Fordelene med designmønstrene erfares best med større systemer som skal vedlikeholdes og videreført over flere år, men kan innebære unødvendig ekstra kode og kompleksitet for mindre programmer i skolesammenheng. Se [t\\_5\\_8\\_3\\_decorator.py](#).

```
if __name__ == "__main__":
    melding = Melding("Dette er en enkel melding.")
    print("Normal melding:", melding.hent_tekst())
    print("Fet melding:", TekstFormatDekorator(melding, "fet").hent_tekst())
    print("Rød melding:", FargeDekorator(melding, "rød").hent_tekst())
    print("Grønn, kursiv melding:", FargeDekorator(
        TekstFormatDekorator(melding, "kursiv"), "grønn").hent_tekst())
    print("Fet, blå melding:", TekstFormatDekorator(
        FargeDekorator(melding, "blå"), "fet").hent_tekst())
```



```
Normal melding: Dette er en enkel melding.
Fet melding: Dette er en enkel melding.
Rød melding: Dette er en enkel melding.
Grønn, kursiv melding: Dette er en enkel melding.
Fet, blå melding: Dette er en enkel melding.
```

## MVC

MVC-mønsteret (*Model-View-Controller*) deler programmet inn i tre hovedobjekter for å gjøre det mer fleksibelt og enklere å gjenbruke. Modellen er selve applikasjonen, visningen står for skjermpresentasjonen, og kontrolleren styrer hvordan brukergrensesnittet (visningen) reagerer på brukerens handlinger.

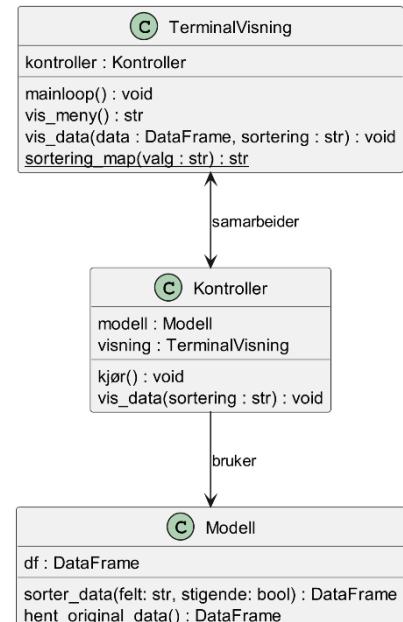
I vårt eksempel består modellen av en liste som kan sorteres på forskjellige måter. Brukgrensesnittet vises i et terminalvindu. Kontrolleren setter i gang en kontinuerlig løkke der brukeren får velge hvordan listen skal sorteres. Basert på brukerens valg, henter kontrolleren de nødvendige dataene fra modellen og instruerer deretter visningen om å vise de sorterte dataene. Se [t\\_5\\_8\\_4\\_mvc\\_cli.py](#).

```
Usorterte/original data:
Navn Alder
Per 23
Kari 34
Ola 45

Vil du fortsette? (ja/nei): 
```

```
Data sortert etter navn (stigende):
Navn Alder
Kari 34
Ola 45
Per 23

Vil du fortsette? (ja/nei): 
```



```
Data sortert etter alder (synkende):
Navn Alder
Ola 45
Kari 34
Per 23

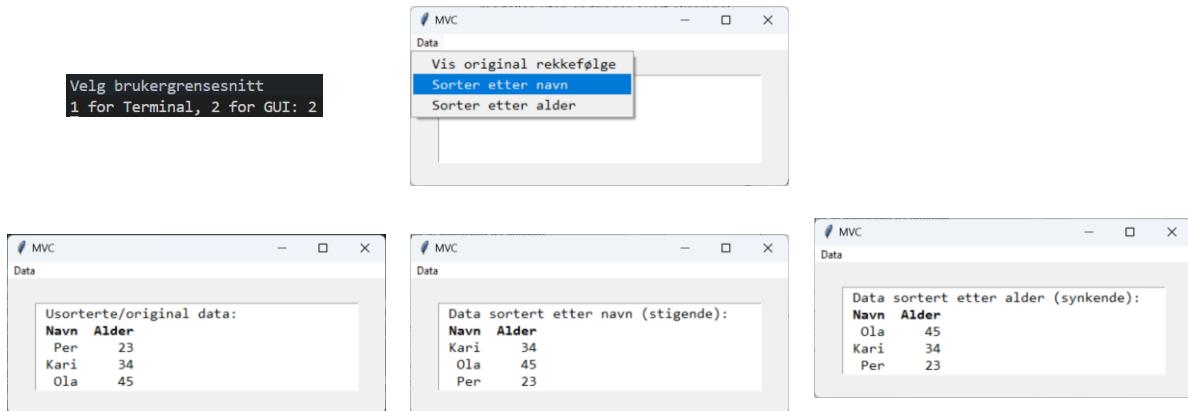
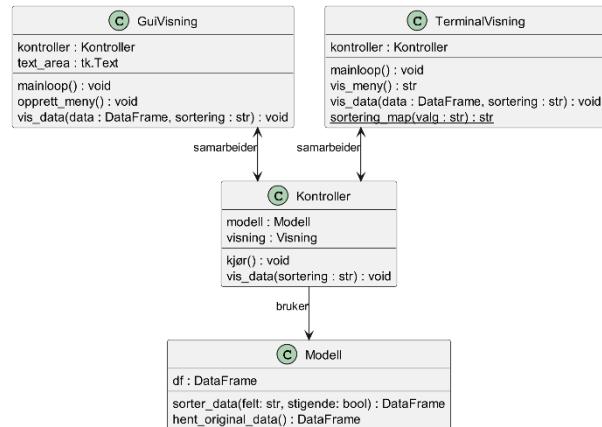
Vil du fortsette? (ja/nei): 
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Hvis vi ønsker å legge til en annen type visning, for eksempel et grafisk brukergrensesnitt, kan vi gjenbruke modellen uten endringer. I vårt eksempel kan brukeren velge mellom terminalvisning eller et grafisk brukergrensesnitt når programmet startes. Ved å navngi hovedløkken `mainloop` i terminalversjonen, som også benyttes av Tkinter i GUI-versjonen, kan kontrolleren også gjenbrukes uendret. Se [t\\_5\\_8\\_4\\_mvc\\_cli\\_og\\_gui.py](#).

Vi kunne haft med en abstrakt klasse `Visning` som krevede at de konkrete visningsklassene implementerte `mainloop`, men vi har valgt å holde det enkelt for å gjøre MVC-mønsteret lettere å forstå.



Det finnes også en annen variant av MVC-mønsteret. Tenk deg at data i modellen oppdateres dynamisk og hyppig fra eksterne kilder, uavhengig av kontrolleren, og vi ønsker at disse endringene automatisk gjenspeiles i visningen. For å oppnå dette introduserer vi en kobling mellom `Visning` og `Modell` ved å bruke *Observer*-mønsteret. Visningen abonnerer på endringer i modellen ved å registrere seg som en observatør. Når modellen oppdateres, varsler den alle registrerte observatører ved å kalle en oppdateringsmetode de har forpliktet seg til å implementere. Dermed henter visningene de nyeste dataene direkte fra modellen når dataene endres.

## 5.9 Hendelser, animasjon, kollisjoner og simulering med tkinter

### Innhold

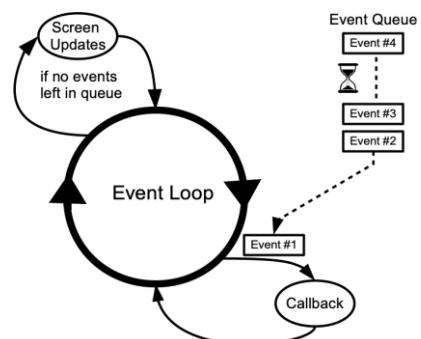
Pygame vs. Tkinter: Programmeringsperspektiv .....	348
Hendelser.....	348
Canvas.....	349
Animasjon.....	350
Objektorientert animasjon .....	351
Kollisjoner.....	353
Simuleringer .....	354

### Pygame vs. Tkinter: Programmeringsperspektiv

- **Hendelseshåndtering:** I Tkinter kobles funksjoner til hendelser via `bind()`. I Pygame henter vi hendelseslisten med `pygame.event.get()`.
- **Animasjon:** Pygame fokuserer på animasjoner med Surfaces og Sprites. Tkinter bruker Canvas for tegning og enkel animasjon.
- **Tidskontroll:** `after()` i Tkinter setter hendelser med tidsforsinkelser. Pygame bruker `clock.tick()` for å styre bilder per sekund.
- **Brukarkomponenter vs. spillelementer:** Tkinter byr på klassiske GUI-komponenter som knapper, tekst- og kombobokser, mens Pygame gjør det enkelt å programmere animasjon og kollisjoner med bruk av sprites.
- **Skjermoppdatering:** administreres automatisk sammen med hendelseshåndtering gjennom `mainloop()` i Tkinter, mens vi må eksplisitt kalle `pygame.display.update()` i en spillokke i Pygame

### Hendelser

I enkle programmer uten grafisk brukergrensesnitt kjøres koden normalt i en sekvens. På den annen side er GUI-programmer hendelsesstyrte. En hendelse oppstår når brukeren utfører en handling som å bevege musen, klikke på en knapp, trykke på en tast, osv. Hendelsene blir plassert i en hendelseskø (event queue) av operativsystemet. Hendelsene er koblet til objektet som utløser hendelsen. Vi bestemmer selv hvilke hendelser som skal behandles. Dette er kjent som å registrere en lytterfunksjon. Deretter tilknytter vi en hendelse til koden som skal utføres når hendelsen oppstår. Vi skriver denne koden i lytterfunksjonen selv. Denne lytterfunksjonen blir også omtalt som *event handler* eller *callback*.



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

I **tkinter** kjører en hendelsessløyfe (*event loop*) som kontinuerlig henter hendelser fra hendelseskøen og utfører de tilknyttede lytterfunksjonene. En lytterfunksjon må utføre oppgavene raskt, siden programmet ikke responderer på andre hendelser mens koden i lytterfunksjonen kjøres. Prosessen starter ved bruk av metoden **mainloop** som viser **widget**-ene på skjermen, initierer hendelsessløyfen og gir brukeren mulighet til å samhandle med programmet. For mer informasjon, se [Event Loop](#).

Vi tilknytter en hendelse til et objekt ved hjelp av metoden **bind** (**bind\_class** eller **bind\_all**). For informasjon om hendelser ([typer](#), [egenskaper](#), taster, osv.), se [tkdocs](#)

```
from tkinter import *

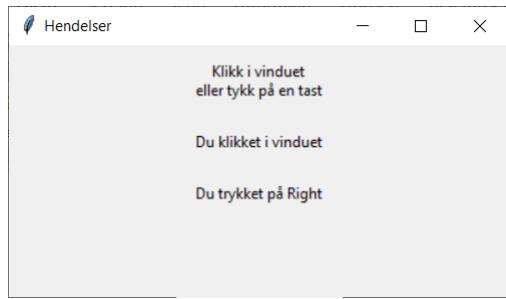
# Lytterfunkjoner
def mus_lytter(event):
    Label(root, text='Du klikket i vinduet').\
        pack(padx=10, pady=10)

def tast_lytter(event):
    tekst = 'Du trykket på ' + event.keysym
    Label(root, text=tekst).\
        pack(padx=10, pady=10)

root = Tk()
root.geometry('400x200')
root.title('Hendelser')
Label(root, text='Klikk i vinduet\neller trykk på en tast').\
    pack(padx=10, pady=10)

# Registrer lytterfunkjoner
root.bind('<Button-1>', mus_lytter)
root.bind('<Key>', tast_lytter)

root.mainloop()
```



Koden som vises ovenfor, finnes i filen [t\\_5\\_9\\_1\\_event\\_tk.py](#).

## Canvas

[Canvas](#) er en **tkinter widget** som fungerer som hvor vi kan tegne elementer. For eksempel kan vi tegne en sirkel med [canvas.create\\_oval\(\)](#). I tillegg til å tegne en ellipse, returnerer metoden også et heltall som representerer en id for oval-elementet på lerretet. Vi kan sammenligne id-en med en indeks i et [list](#)-objekt, men det er en viktig forskjell. Vi kan kun få tilgang til oval-elementet via canvas-objektet og den tilhørende id-en. Det er imidlertid ikke mulig å få en direkte referanse til selve oval-elementet. Dersom vi sletter canvas, vil også oval-elementet bli fjernet. Dersom vi har en referanse (variabel som peker på) et element i et [list](#)-objekt, vil elementet fortsatt eksistere selv om list-objektet fjernes<sup>35</sup>. Canvas **widget**-en inneholder [metoder](#) som vi kan benytte. Her bruker vi metoden [move\(\)](#) for å flytte ballen.

Se [t\\_5\\_9\\_2\\_flytt\\_ball\\_tk.py](#).

<sup>35</sup> Se [Forhold mellom klasser](#) i 3.5 Klassediagram og [garbage collector](#).

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
import tkinter as tk

def flytt(event):
    key = event.keysym
    match key:
        case "Right": canvas.move(id, 10, 0)
        case "Left": canvas.move(id, -10, 0)

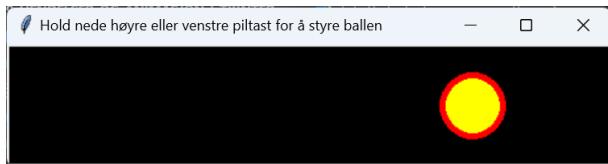
root = tk.Tk()
root.geometry("400x100")
root.title("Flytt ball med høyre og venstre piltast")
canvas = tk.Canvas(root, width=400, height=100, bg="black")
canvas.pack()
id = canvas.create_oval(175, 25, 225, 75, outline="red",
                       fill="yellow", width=5)
root.bind("<Key>", flytt)
root.mainloop()
```



### Animasjon

I [øving 3.3.2](#) brukte vi `time.sleep()` for å lage animasjonen. Dette er imidlertid ikke en optimal tilnærming, da programmet ikke reagerer på andre hendelser mens det "sover". En bedre metode i `tkinter` er å bruke `after()`. Denne funksjonen er tidsstyrт og kaller opp en funksjon etter et angitt antall millisekunder, som lar programmet også håndtere andre hendelser i mellomtiden. I vårt eksempel flytter vi ballen 5 piksler enten til høyre eller venstre hver 40. millisekund, basert på hvilken tast som er trykket ned. Dette gir en hastighet på 125 piksler per sekund. Funksjonen sørger også for at ballen holdes innenfor canvasgrensene ved å justere dens posisjon om nødvendig. Vi bruker `move`, som gir en relativ bevegelse, i stedet for `moveto`, som ville gitt en absolutt bevegelse. Se [t\\_5\\_9\\_3\\_styrt\\_animasjon\\_tk.py](#).

```
def animér():
    if "Right" in pressed_keys:
        canvas.move(ball_id, dx, 0)
    if "Left" in pressed_keys:
        canvas.move(ball_id, -dx, 0)
    # Hold ballen innenfor canvas
    coords = canvas.coords(ball_id)
    if coords[0] < 0:
        canvas.coords(ball_id, 0, 25, 50, 75)
    elif coords[2] > 500:
        canvas.coords(ball_id, 450, 25, 500, 75)
    root.after(40, animér)
```

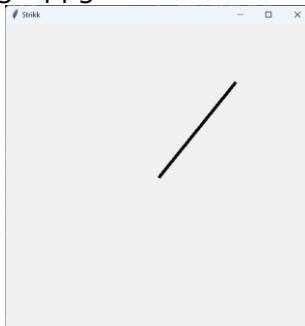


I det følgende eksempelet bruker vi hendelsen som triggas når vi beveger musen. Hendelsen heter '[<Motion>](#)', og hendelseobjektet, som vi har kalt `event`, har med seg x- og y-verdiene til musa. Disse verdiene bruker vi til å oppdatere en rett linje fra midten av lerretet til muspekerens posisjon, noe som skaper en slags "strikk"-effekt. Vanlige venstreklikk på musa, kalt '[<Button-1>](#)', gir også x- og y-verdiene fra stedet der musa befant seg da den ble klikket på. Disse verdiene skal vi bruke i øvingsoppgavene. Se [t\\_5\\_9\\_4\\_strikk\\_tk.py](#).

```
import tkinter as tk

def strikk(event):
    canvas.coords(id, 250, 250, event.x, event.y)

root = tk.Tk()
root.geometry("500x500")
root.title("Strikk")
canvas = tk.Canvas(root, width=500, height=500)
canvas.pack()
id = canvas.create_line(250, 250, 250, 250,
                       fill="black", width=5)
canvas.bind("<Motion>", strikk)
root.mainloop()
```



Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

*Tweening* er en teknikk som brukes i animasjon for å skape glatte overganger mellom nøkkelbilder ved å interpolere verdier som posisjon, farge, rotasjon og størrelse.

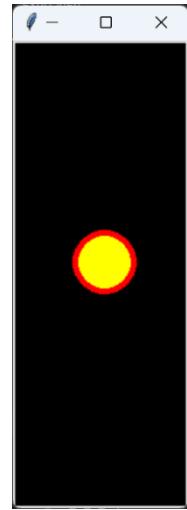
Overgangene blir mer naturlige når de ikke er lineære. Tenk på en bil som akselererer fra 0 til 80 km/t. Det tar tid både å nå toppfarten og å senke farten igjen. Dette kalles for *ease in* og *ease out* i *tweening*-

sammenheng. Denne [YouTube videoen](#) sammenligner ulike *easing* typer. Det finnes *tweening* bibliotek for Python, men vi må gjøre mye selv.

```
import tkinter as tk
import pytweening as tween

WIDTH, HEIGHT = 150, 400
FPS = 24
BILDER = FPS * 2 # Tweening i 2 sekunder
INTERVALL = 1000 // FPS
AVSTAND = HEIGHT - 60
x = (WIDTH - 50) // 2
DIAMETER = 50

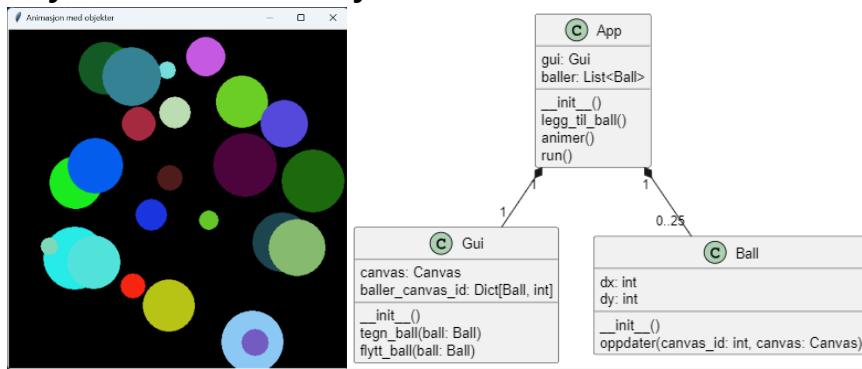
def animerteller):
    if teller <= BILDER:
        y = AVSTAND * tween.easeOutBounce(teller / BILDER)
    else:
        y = AVSTAND
        if teller > FPS * 3:
            teller = 0
            y = 0
    canvas.moveto(ball_id, x, y)
    root.after(INTERVALL, lambda: animerteller + 1))
```



`pip install pytweening --use-pep517`

Her bruker vi `easeOutBounce` for å få en ball til å sprette noen ganger når den treffer bakken. Først bestemmer vi varigheten av animasjonen for å vite antall bilder. Deretter bruker vi en teller som argument for å indikere prosentandelen av fullført animasjon. Returverdien ligger også mellom 0 og 1, men den blir *tweenet*. Se [t\\_5\\_9\\_5\\_tweening\\_tk.y](#).

### Objektorientert animasjon



Det er også mulig å lage objektorienterte animasjoner. I dette eksempelet oppretter vi en ny ball hvert annet sekund inntil det er 25 baller på skjermen. Hver av ballene tildeles vilkårlig størrelse, farge,

utgangsposisjon og hastighet. Vi bruker tre klasser: `App`, `Gui` og `Ball`. `App`-instansen styrer animasjonen ved å opprette og oppdatere `Ball`-instansene, mens `Gui`-instansen håndterer brukergrensesnittet. Forholdet mellom disse klassene refereres til som komposisjon, og det skiller seg fra arv. `Ball`-klassen arver ingen attributter eller metoder fra `App`- eller `Gui`-klassene. `App`-instansen har imidlertid tilgang til `Ball`-instansene gjennom referanser som ligger lagret i listen `baller`. Se [t\\_5\\_9\\_6\\_objekter\\_tk](#).

Programmet startes ved å opprette en instans av klassen `App`. Denne `if`-setningen er tatt med for å unngå at programmet starter automatisk dersom modulen (`.py`-filen) blir importert i andre programmer.

```
if __name__ == "__main__":
    app = App()
    app.run()
```

Vurderingsseksempler

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Konstruktøren `__init__` i `App`-klassen oppretter en `Gui`-instans og en tom liste for `Ball`-objekter. Deretter opprettes en ny ball og animasjonen settes i gang.

`Gui` er en underklasse av `Tk`-klassen. Derfor tilsvarer `self` i `Gui`-objektet `root` i de tidligere nevnte programmene. Konstruktøren starter med å kalle konstruktøren til overklassen `Tk`. Deretter tildeles vinduet en tittel, og det blir opprettet en `Canvas-widget`.

Metoden `legg_til_ball` i `App`-klassen oppretter en ny ball, og legger deretter referansen til denne til listen `baller` ved hjelp av `append()`. Deretter brukes metoden `Gui.tegn_ball` for å opprette ballen ballen på `canvas`.

Hvis antallet baller i listen er mindre enn 25, vil `legg_til_ball()` bli kalt igjen etter en ventetid på 2 sekunder ved hjelp av `after()`.

Når vi oppretter en ny instans av `Ball`-klassen, blir konstruktøren kjørt. `__init__` gir ballen en hastighet.

`Gui.tegn_ball()` tar et `Ball`-objekt som argument og legger det ut på `canvas`. Metoden genererer vilkårlige verdier for diameter, farge og startposisjon (`x` og `y`) før den tegner ballen på canvas. `canvas_id` som returneres av `create_oval` lagres i en ordbok i brukergrensesnittet med `Ball`-objektet som nøkkel for senere referanse.

Diameter, farge og startposisjon (`x` og `y`) settes til vilkårlige verdier. Det er totalt  $2^{24}$  (16 777 216) RGB fargekombinasjoner. `randint` genererer en tilfeldig verdi blant disse alternativene som et desimaltall. Denne verdien blir deretter konvertert til heksadesimalformat som begynner med prefikset '`Ox`'. Dette prefikset fjernes ved å bruke *slicing* `[2:]`. Dersom tallet inneholder færre enn 6 tegn, fyller `zfill(6)` på med nuller foran. '`#`' skjøtes så med denne hex-teksten, og vi får et fargeformat som kan brukes i `create_oval`.

Metoden `App.animer` blir kalt igjen hvert 40 millisekund. Den går gjennom alle ballene i listen `baller` og kaller `Gui.flytt_ball()` for hver ball.

`Gui.flytt_ball()` bruker ballens `canvas_id` fra ordboken til å kalle ballens `oppdater()`.

`Ball.oppdater()` henter ballens posisjon på `canvas` og sjekker om neste posisjon vil bevege seg utenfor `canvas`. I så fall snus bevegelsesretningen. Til slutt flyttes ballen på `canvas`.

Vi kunne ha tegnet alt på nytt for hvert bilde slik som i *pygame*, men i stedet for å kalle `create_oval` for hver ball ved hver ny visning, bruker vi `move` for å flytte ballene. Dette er

```
class App:  
    def __init__(self):  
        self.gui = Gui()  
        self.baller = []  
        self.legg_til_ball()  
        self.animer()
```

```
class Gui(tk.Tk):  
    def __init__(self):  
        super().__init__()  
        self.title("Animasjon med objekter")  
        self.canvas = tk.Canvas(self, bg="black",  
                               width=BREDDE, height=HØYDE)  
        self.canvas.pack()
```

```
def legg_til_ball(self):  
    if len(self.baller) < BALLER:  
        ball = Ball()  
        self.baller.append(ball)  
        self.gui.tegn_ball(ball)  
        self.gui.after(2000, self.legg_til_ball)
```

```
class Ball:  
    def __init__(self):  
        self.dx = choice([-1, 1]) * randint(5, 10)  
        self.dy = choice([-1, 1]) * randint(5, 10)
```

```
def tegn_ball(self, ball):  
    diameter = randint(25, 100)  
    farge = "#" + \  
           hex(randint(0, 2**24 - 1))[2:].zfill(6)  
    x = randint(0, BREDDE - diameter)  
    y = randint(0, HØYDE - diameter)  
    canvas_id = self.canvas.create_oval(  
        x, y, x + diameter, y + diameter,  
        fill=farge, outline="")  
    self.baller_canvas_id[ball] = canvas_id
```

```
def animer(self):  
    for ball in self.baller:  
        self.gui.flytt_ball(ball)  
    self.gui.after(INTERVALL, self.animer)
```

```
def flytt_ball(self, ball):  
    canvas_id = self.baller_canvas_id[ball]  
    ball.oppdater(canvas_id, self.canvas)
```

```
def oppdater(self, canvas_id, canvas):  
    x1, y1, x2, y2 = canvas.coords(canvas_id)  
    if x1 + self.dx < 0 or x2 + self.dx > BREDDE:  
        self.dx *= -1  
    if y1 + self.dy < 0 or y2 + self.dy > HØYDE:  
        self.dy *= -1  
    canvas.move(canvas_id, self.dx, self.dy)
```

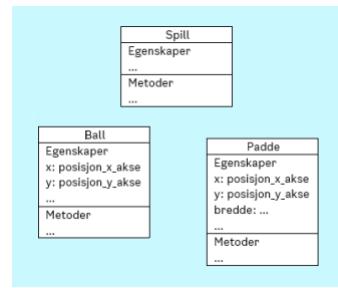
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

enklere og mer effektivt. For å oppnå dette uten å lagre `canvas_id` i `Ball`-objektene, bruker vi en ordbok i `Gui`-klassen. Denne ordboken kobler hver ball til dens `canvas_id`, noe som holder GUI-informasjon separat fra modellen. Det gjør det også mulig å detektere kollisjoner ved hjelp av metoden `find_overlapping`, som forklart i avsnittet om kollisjoner i neste avsnitt.

I vårt eksempel lagrer vi kun hastigheten som et attributt i `Ball`-objektene. Ideelt sett burde vi også lagret størrelse, farge og posisjon i `Ball`-objektene for å følge prinsippet om at all relevant informasjon skal ligge i modellen. Men siden vi ikke har behov for denne informasjonen i modellen etter at ballene er opprettet, tildeles disse verdiene til oval-elementet på `canvas` for visningsformål, uten å opprette en lokal kopi i modellen. Dette gjør at vi unngår unødvendig kompleksitet i modellen, ettersom farge og størrelse forblir uendret, mens posisjonen oppdateres relativt med `move` og hastighet.

Hvorfor oppretter vi våre egne `Ball`-objekter når vi tidligere klarte oss med oval-elementene på lerretet? En av grunnene er at vi ønsker å lære objektorientert programering og implementere våre egne klasser, i tråd med kompetansemålene i læreplanen for IT 2 og eksamensoppgaver som krever dette. Se eksempelvis klassediagrammet til MultiPong-oppgaven. Når noe ikke lar seg modellere som et tall eller en tekst, kan det være hensiktsmessig å bruke en klasse. Attributter og metoder som har en naturlig sammenheng, grupperes i en klasse. Det gjør koden mer strukturert og lettere å vedlikeholde.



En annen tilnærming ville vært å tegne alt på nytt for hver oppdatering, og ikke bruke `canvas_id` i det hele tatt. I stedet kunne vi ha brukt en liste eller ordbok for å holde styr på alle egenskaper til ballene, som hastighet, posisjon, størrelse og farge. Disse kunne vi brukt til å tegne ballene med `create_oval`. Samtidig som dette ville være mer komplisert og mindre effektivt, ville en slik tilnærming heller ikke dra nytte av bedre organisering, enklere vedlikehold og økte gjenbruksmuligheter vi oppnår med objektorientering.

En tredje grunn kan være betydningen av å separere modellen fra brukergrensesnittet, noe som kan oppnås ved å bruke [Model-view-controller \(MVC\)](#)-designmønsteret. Se [MVC, 5.8 Designmønstre](#)

### Kollisjoner

Når vi utvikler animasjoner, oppstår ofte behovet for å sjekke for kollisjoner. Ved å spore `canvas`-elementenes `id`, kan vi benytte metoden `find_overlapping`. Denne metoden returnerer en tuppel med `id`-en til alle elementene som overlapper med et gitt rektangel, noe som gjør kollisjonsdeteksjonen enklere og mer effektiv. Koden i bildet til over til høyre viser hvordan dette gjøres i et løsningsforslag til MultiPong laget med `tkinter`. Se [t\\_5\\_6\\_7\\_multipong\\_tk.py](#).

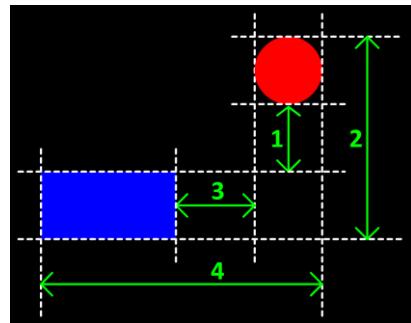
```
def sjekk_kollisjoner(self):
    padde_coords = self.canvas.coords(self.padde.id)
    overlaps = self.canvas.find_overlapping(*padde_coords)
    for ball in self.baller:
        if ball.id in overlaps:
            ball.speed[1] *= -1
            self.canvas.move(ball.id, 0, -20)
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Dersom vi skal utføre denne sjekken manuelt, må vi forsikre oss om at

1. bunnen på den røde ballen ligge under toppen på den blå padden
2. toppen på den røde ballen er over bunnen på den blå padden
3. venstre side på den røde ballen ligge til venstre for den høyre siden på den blå padden
4. høyre side på den røde ballen ligge til høyre for den venstre siden på den blå padden.



## Simuleringer

Pygame er godt egnet for programmering av animasjoner og kollisjoner med *sprites*, men til IT-2-eksamen våren 2024 kunne vi klart oss med *tkinter*. Som vi nettopp så, kunne vi brukt det til å kode MultiPong<sup>36</sup> også.

Hovedoppgaven om cellenes livssyklus (*Game of Life*) løste vi i [oppgave 4.4.5](#) med pygame. Her er en alternativ løsning med *tkinter*, som tilbyr tilstrekkelig funksjonalitet for å håndtere denne typen simuleringer.

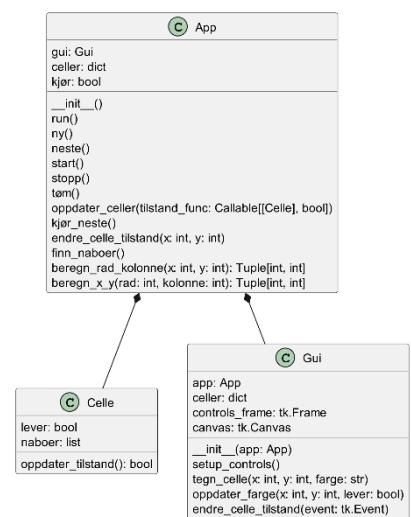
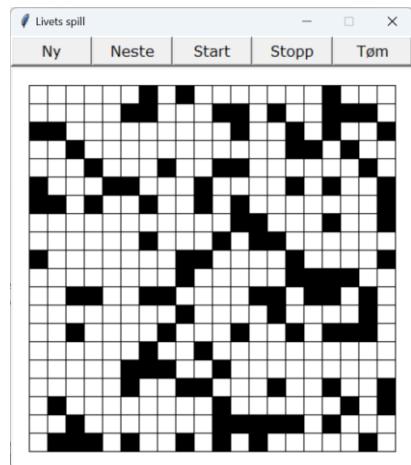
Se [t\\_5\\_9\\_8\\_livets\\_spill\\_tk.py](#) for den fullstendige koden.

Øverst har vi en knapperad lik den i [Tkinter App-GUI malen](#) i [3.9 Reelle datasett med OOP og GUI](#) Rutenettet tegnes på canvas ved å sette sammen rektangler med svarte kantlinjer. Programmet består av tre klasser: *App*, *Celle* og *Gui*.

*Celle*-objektene har to attributter: *lever*, som er *True* eller *False*, og *naboer*, som er en liste med naboceller. Den har også en metode, *oppdater\_tilstand*, som finner ut om den skal leve i neste generasjon eller ikke.

*App*-objektet oppretter *Gui*-instansen og alle cellene, som plasseres i en ordbok med (rad, kolonne)-tupler som nøkkel og *Celle*-instansene som verdi. Attributtet *kjør* blir brukt til å animere generasjonene fortløpende. Metodene er som følger:

- *ny()* er koblet til Ny-knappen og setter hver celle til levende med 1/3 sannsynlighet.
- *nest()* er koblet til Neste-knappen og viser neste generasjon.
- *start()* er koblet til Start-knappen og setter i gang animasjon og viser generasjonene fortløpende.



<sup>36</sup> [Eksemploppelgaver til IT-2 eksamen](#), høsten 2022.

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

- `stopp()` er koblet til Stopp-knappen og stopper animasjonen.
- `tøm()` er koblet til Tøm-knappen og setter alle cellene til å være døde.
- `oppdater_celler()` brukes av `ny()`,  `neste()` og `tøm()` for å oppdatere lever-attributtet til alle cellene med en funksjon som tar en celle som argument. Celler som blir endret, oppdateres i `Gui`-instansen.
- `kjør_neste()` blir brukt av `start()` for å kjøre animasjonen.
- `endre_celle_tilstand()` endrer lever-attributtet til en celle som blir klikket på.
- `finn_naboer()` kjøres én gang etter at alle cellene er opprettet. Hver celle får en liste med sine naboceller.
- `beregn_rad_kolonne()` tar x- og y-verdiene fra `canvas` og returnerer `rad` og `kolonne`.
- `beregn_x_y()` tar rad- og kolonne-verdiene i `App`-instansen og returnerer x- og y-verdiene for `Gui`-instansen.

I brukergrensesnittet blir hver celle tegnet som et rektangel på `canvas`. Id-en til cellen blir tatt vare på som verdi i en ordbok, `celler`, som har  $(x, y)$ -tuplene som nøkkel. Dette gjør det enkelt å finne rektanglet når fargen skal oppdateres, i stedet for å tegne et nytt rektangel oppå det gamle og fylle opp `canvas`.

- `setup_controls()` setter opp knappene.
- `tegn_celle()` tegner en hvit celle med svart kant i posisjon `x, y` og tar vare på id-en i `celler`-ordboken.
- `oppdater_farge()` finner id-en basert på `x, y` og oppdaterer fargen til cellen.
- `endre_celle_tilstand()` kaller tilsvarende funksjon i `App`-instansen med `x, y`-argumenter til musepekerens posisjon da den ble klikket på.

Vurderingsseksemplar

## 5.10 Geodata

### Innhold

Innledning .....	356
Geopy.....	357
Geopandas .....	357
Byer i Norge.....	359
Webbaserte kart- og navigasjonsapplikasjoner.....	360
OpenStreetMap.....	361
OSMnx .....	361
OSMnx og GeoPandas (lsbreer).....	362
Folium .....	362
Øvingsoppgaver.....	363

### Innledning

Geodata, eller geografiske data, er en viktig ressurs for å forstå og analysere geografiske fenomener og mønstre i samfunnet.

Geodata omfatter en rekke forskjellige typer informasjon om steder og ting rundt oss, som for eksempel

- naturressurser som elver, innsjøer, skoger og fjell
- infrastruktur som veier, jernbaner og flyplasser
- offentlige rom som parker, torg og strender
- konstruksjoner som broer, tårn og vindmøller,
- bygninger som skoler, sykehus og svømmehaller



Geodata inkluderer ofte målinger tatt på spesifikke steder, som for eksempel temperatur, nedbørsmengde og luftkvalitet. For å beskrive nøyaktig hvor et bestemt sted er, benyttes ulike koordinatsystemer. Vi vil holde oss til de tradisjonelle bredde- og lengdegradene, og de Python-pakkene vi skal bruke, er vanligvis i stand til å håndtere geodatafilene riktig, uansett hvilket koordinatsystem de er basert på.

Geografiske informasjonssystemer (GIS) er verktøy som brukes for å samle, lagre, analysere og presentere geografisk informasjon. Offentlige og private organisasjoner bruker avanserte

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

GIS-systemer som [ArcGIS](#), [QGIS](#) og [MapInfo](#) til å håndtere store datamengder, analysere komplekse sammenhenger og ta avgjørelser basert på geografisk informasjon. Myndighetene bruker GIS til å planlegge infrastruktur som veier og jernbaner, og til å forvalte naturressurser som skog og vann. Bedrifter kan bruke geodata til å analysere markeder, optimalisere logistikk og planlegge nye lokasjoner.

Selv om vi ikke skal bruke de avanserte GIS-systemene nevnt ovenfor, vil vi bli kjent med noen GIS-biblioteker som kan være svært nyttige for å utforske og visualisere geografisk data.

### Geopy

[geopy \(docs\)](#) er et Python-bibliotek som brukes til geografiske beregninger og geokoding. Det lar oss enkelt arbeide med geografiske koordinater, avstander og steder. For eksempel kan vi bruke [geopy](#) til å finne koordinatene (bredde- og lengdegrad) til Oslo ved å sende "Oslo" som en adresse til [geocode](#)-metoden og hente ut koordinatene fra svaret. [geopy](#) fungerer som et grensesnitt som kobler oss til flere geokodingstjenester i én enkelt pakke, uten å tilby egen geokodingstjeneste. Her bruker vi [Nominatim](#), som er et API for å søke i OpenStreetMap-data.

[geopy](#) kan også brukes til å beregne avstander mellom steder, og det tar hensyn til jordas krumning. Her beregner vi avstanden mellom Oslo og Bergen i luftlinje med [distance](#)-metoden. Husk [pip install geopy](#).

```
1 from geopy.geocoders import Nominatim
2
3 # Opprett en geokoder
4 geolocator = Nominatim(user_agent="min_agent")
5
6 # Finn koordinatene til Oslo
7 location = geolocator.geocode("Oslo")
8 oslo_B = location.latitude
9 oslo_L = location.longitude
10 print(f"Koordinater for Oslo: {oslo_B} N, {oslo_L} Ø")
   ✓ 0.2s
Koordinater for Oslo: 59.9133301 N, 10.7389701 Ø
```

```
1 from geopy.distance import distance
2
3 # Koordinater for Oslo og Bergen
4 oslo = (59.9133301, 10.7389701)
5 bergen = (60.3943055, 5.3259192)
6
7 # Beregn avstand mellom Oslo og Bergen
8 avstand_oslo_bergen = distance(oslo, bergen).km
9 print(f"Avstanden mellom Oslo og Bergen er {avstand_oslo_bergen:.0f} km")
10
   ✓ 0.0s
Avstanden mellom Oslo og Bergen er 305 km
```

Denne funksjonaliteten kan være nyttig i mange GIS-relaterte oppgaver. Vær klar over at [geopy](#) krever internettforbindelse, da det benytter seg av online geokodingstjenester for å hente geografiske data. Dette betyr at vi må ha en aktiv internettforbindelse for å bruke [geopy](#) i våre Python-programmer.

### Geopandas

[Geopandas](#) er et populært bibliotek i Python for å plotte egne kart og håndtere geografiske data. Det bygger på Pandas og gjør det enkelt å arbeide med geografiske data sammen med annen tabellbasert data. En av de viktigste kolonnene i en Geopandas GeoDataFrame er [geometry](#), som inneholder selve geometrien til geografiske objekter, som punkter, linjer eller polygoner. [Geopandas](#) støtter forskjellige filformater som brukes av GIS-systemer, som for eksempel shapefiler (SHP), Geodatabase (GDB), Geopackage (GPKG) og GeoJSON (JSON eller GEOJSON). Selv om hovedformålet med [Geopandas](#) er å plotte kart, kan det også brukes til å håndtere geografiske data og utføre romlig analyse.

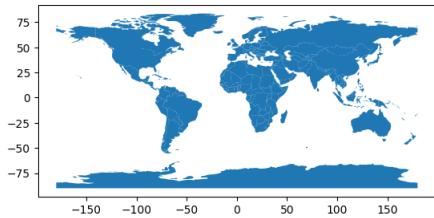
[GeoPandas](#) kommer med tre ferdiglagde kart, [naturlearth\\_cities](#), [naturlearth\\_lowres](#), og [nybb](#). Når de er lastet inn som [GeoDataFrames](#), inneholder disse kartene punkter for flere større byer eller lavoppløselige polygoner for grensene til alle land. For å plotte kart bruker [Geopandas Matplotlib](#). I GeoPandas 1.0 vil disse kartene bli fjernet. Nedenfor følger noen

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

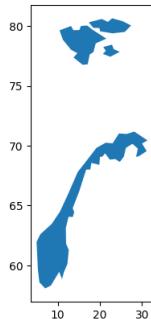
eksempler med `naturalEarth_lowres` som vi har lastet ned fra [www.naturalearthdata.com](http://www.naturalearthdata.com) i stedet. Først hele verden:

```
import geopandas as gpd
import matplotlib.pyplot as plt
import os
from pathlib import Path
sti = Path(os.getcwd())
fil = sti.joinpath("ne_110m_admin_0_countries", "ne_110m_admin_0_countries.shp")
world = gpd.read_file(fil)
world.plot()
plt.show()
```



Så bare Norge:

```
# Velg og vis bare Norge
norway = world[world.SOVEREIGNT == "Norway"]
norway.plot()
plt.show()
```



Hvis vi velger ut bare Norge, blir det i første omgang feil i forholdet mellom x- og y-aksen, men det kan vi rette på med `figsize` og `ax.set_aspect("auto")`. For å få med bare Fastlands-Norge, kan vi benytte `ax.set_xlim` og `ax.set_ylim`.

```
# Plot Norge uten Svalbard
import geopandas as gpd
import matplotlib.pyplot as plt

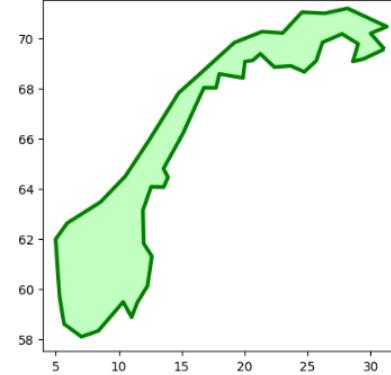
# Last inn verden
world = gpd.read_file(gpd.datasets.get_path("naturalEarth_lowres"))

# Filterer ut Norge
norway = world[world.name == "Norway"]

# Plot Norge uten Svalbard
fig, ax = plt.subplots(figsize=(5,5))

norway.plot(ax=ax, facecolor="#c0ffcc", edgecolor="green", linewidth=3)
ax.set_aspect("auto")
ax.set_xlim(4, 32)
ax.set_ylim(57.5, 71.5)

plt.show()
```



[GeoJSON](#) er et format for kodding av forskjellige geografiske datastrukturer og er spesifisert i [RFC 7946](#). Når vi oppretter en `GeoDataFrame`, kan vi lese inn et GeoJSON-objekt eller bruke en tilsvarende ordbok (`dict`) eller objekt som kilde. Her er et eksempel på hvordan en `GeoDataFrame` for Oslo kan opprettes og plottes oppå en kartfigur for Norge uten Svalbard: Vi lager en `GeoDataFrame` for `oslo` og bruker en `GeoJSON`-datastruktur med to nøkler, `name` og `geometry`, som tilsvarer kolonnene i `GeoDataFrame`-tabellen. `crs=world.crs` blir brukt til å angi det samme koordinatsystemet (*Coordinate Reference System, CRS*) for "`oslo`" som er brukt i `world` `GeoDataFrame`-tabellen. Dette er viktig for å sikre at de geografiske dataene i `oslo` blir riktig projisert og sammenlignet med andre geodata som er i samme CRS som `world`.

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

```
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt

# Last inn verden
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))

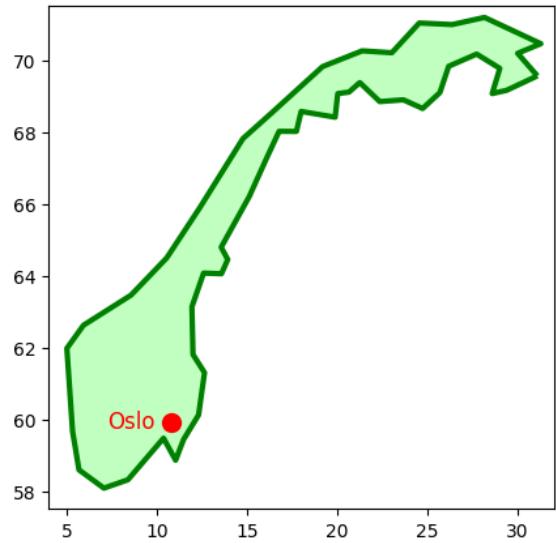
# Filter ut Norge
norway = world[world.name == "Norway"]

# Lag en GeoDataFrame med et punkt for Oslo
oslo = gpd.GeoDataFrame({"name": ["Oslo"], "geometry": [
    Point(10.75, 59.91)]}, crs=world.crs)

# Plot Norge uten Svalbard
fig, ax = plt.subplots(figsize=(5,5))
norway.plot(ax=ax, facecolor="#c0ffc0", edgecolor="green", linewidth=3)
ax.set_aspect('auto')
ax.set_xlim(4, 32)
ax.set_ylim(57.5, 71.5)

# Plot Oslo oppå Norge
oslo.plot(ax=ax, marker="o", color="red", markersize=100)
for x, y, label in zip(oslo.geometry.x, oslo.geometry.y, oslo.name):
    ax.annotate(label, xy=(x, y), xytext=(-35, -3),
                textcoords="offset points", size=12, color="red")

plt.show()
```



Vi kan også bruke mer detaljerte kartdata, som for eksempel [countries.geojson](#) fra <https://datahub.io/core/geo-countries>

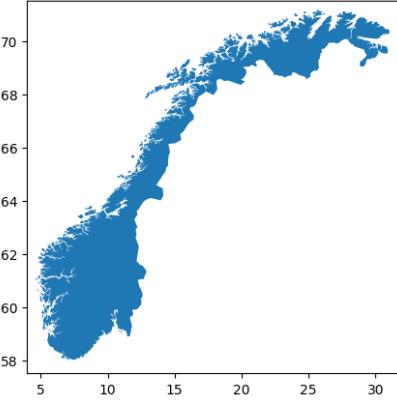
```
import geopandas as gpd
import matplotlib.pyplot as plt

# Hent verdenskartet fra datahub.io
# url = "https://datahub.io/core/geo-countries/r/countries.geojson"
# verden = gpd.read_file(url)

# Bruk en nedlastet versjon av verdenskartet
verden = gpd.read_file('countries.geojson')

# Filter ut Norge
norge = verden[verden['ADMIN'] == 'Norway']

# Plott Norge
fig, ax = plt.subplots(figsize=(5,5))
norge.plot(ax=ax)
ax.set_aspect('auto')
ax.set_xlim(4, 32)
ax.set_ylim(57.5, 71.5)
plt.show()
```



### Byer i Norge

La oss plotte alle byene på norgeskartet. Vi kan kopiere alle bynavnene fra [Wikipedia](#), eller vi kan bruke [BeautifulSoup](#) til å skrape websiden. Vi ble kjent med [BeautifulSoup](#) i [5.6](#)

[Etterligning \(mocking\) med pytest](#) da vi [testet innholdet på en webpage](#). Her velger vi riktig tabell og leser alle radene. I hver rad plukker vi ut byen som står i første kolonne.

```
1 # Finn listen over byer i Norge på Wikipedia
2 import requests
3 from bs4 import BeautifulSoup
4
5 # Send en forespørsel til Wikipedia-siden
6 url = 'https://no.wikipedia.org/wiki/Lista_over_norske_byer'
7 response = requests.get(url)
8
9 # Analyser HTML-innholdet ved hjelp av Beautiful Soup
10 soup = BeautifulSoup(response.content, 'html.parser')
11
12 # Finn tabellen med byene
13 table = soup.find_all('table')[1]
14
15 # Hent byene fra tabellradene
16 byer = []
17 for row in table.find_all('tr'):
18     cols = row.find_all('td')
19     if len(cols) > 0:
20         by = cols[0].text.strip()
21         byer.append(by)
22
23 # Skriv ut listen med bynavnene
24 display(byer)
```

```
['Alta',
'Arendal',
'Askim',
'Bardufoss',
'Bergen',
'Bodø',
'Brekstad',
'Brevik',
'Brumunddal',
'Bryne',
'Bronnaysund',
'Drammen',
'Drobak',
'Egersund',
'Elverum',
'Fagernes',
'Farsund',
'Fausko',
'Finnsnes',
'Flekkfjord',
'Florø',
'Fosnavåg',
'Fredrikstad',
'Ferde',
```

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

Når vi har samlet alle byene i en liste, kan vi bruke [geopy](#) til å finne bredde- og lengdegradene, noe som kan ta omtrent ett minutt. Disse tar vi vare på i en [pandas DataFrame](#).

```
# Finn breddegrad og lengdegrad for hver by
import pandas as pd
from geopy.geocoders import Nominatim

# Lag en geolocator som kan brukes til å finne breddegrad og lengdegrad
geolocator = Nominatim(user_agent="min_agent")

# Definer en funksjon som finner breddegrad og lengdegrad for en by

def get_lat_lon(by: str) -> tuple:
    location = geolocator.geocode(f'{by}, Norway')
    if location is not None:
        return (location.latitude, location.longitude)
    else:
        return (None, None)

# Initialiser lister for breddegrad og lengdegrad
breddegrader = []
lengdegrader = []

# Finn breddegrad og lengdegrad for hver by
for by in byer:
    lat, lon = get_lat_lon(by)
    breddegrader.append(lat)
    lengdegrader.append(lon)

# Opprett en DataFrame med bynavn, breddegrad og lengdegrad
df = pd.DataFrame({'By': byer, 'Breddegrad': breddegrader, 'Lengdegrad': lengdegrader})

# Vis alle data i df
# pd.set_option('display.max_rows', None)
display(df)
```

	By	Breddegrad	Lengdegrad
0	Alta	69.966605	23.273328
1	Arendal	58.426730	8.946949
2	Askim	59.583222	11.164191
3	Bardufoss	69.065637	18.515924
4	Bergen	60.394306	5.325919
5	Bodø	67.309478	13.915442
6	Brekstad	63.686872	9.666500
7	Brevik	59.052889	9.699978
8	Brumunddal	60.884916	10.941839
9	Bryne	58.735525	5.547818
10	Brønnøysund	65.474875	12.211639
11	Drammen	59.696494	10.175642
12	Dørbak	59.663300	10.629339
13	Egersund	58.451512	6.000755
14	Elverum	60.972802	11.734442
15	Fagernes	60.986098	9.236413
16	Farsund	58.038737	6.584827
17	Fauske	67.236715	15.784422
18	Finnsnes	69.229602	17.791297
19	Flekkefjord	58.294936	6.612963
20	Florø	61.600258	5.034960
21	Fosnavåg	62.342106	5.634243
22	Fredrikstad	59.213361	10.936160
23	Førde	61.452176	5.857172
24	Gjøvik	60.872162	10.503886

Nå kan vi plotte lokasjonen til byene på kartet av Norge. Først oppretter vi en [GeoDataFrame](#) fra dataene i [pandas](#)-tabellen. Deretter plotter vi den på kartet av Norge uten Svalbard.

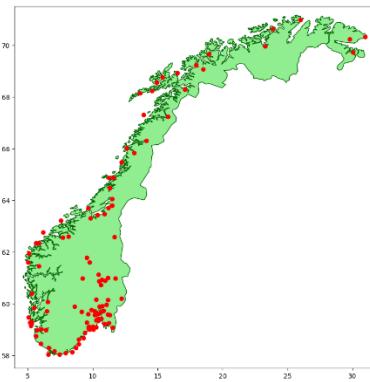
```
# Plott kartet med byene
import geopandas as gpd
import matplotlib.pyplot as plt

# Hent verdenskartet
# https://datahub.io/core/geo-countries#data-climate
world = gpd.read_file('countries.geojson')

# Filter ut Norge
norway = world[world['ADMIN'] == 'Norway']

# Plott Norge uten Svalbard
fig, ax = plt.subplots(figsize=(10, 10))
norway.plot(ax=ax, color='lightgreen', edgecolor='darkgreen')
# Plott byene
gdf = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(
    df.Lengdegrad, df.Breddegrad))
gdf.plot(ax=ax, color='red')

# Juster kartet
ax.set_aspect('auto')
ax.set_xlim(4, 32)
ax.set_ylim(57.5, 71.5)
plt.show()
```



## Webbaserte kart- og navigasjonsapplikasjoner

I 2023 er det flere populære webbaserte kart- og navigasjonsapplikasjoner som er mye brukt over hele verden. Noen av de mest brukte er [Google Maps](#), [Google Maps](#), Apple Maps, [Bing Maps](#) og [OpenStreetMap](#). Apple Maps er ikke tilgjengelig som en egen nettleserversjon, men er laget for bruk på Apple-enheter. Det er også mange andre mindre kjente webbaserte kartapplikasjoner som er populære i visse regioner eller for spesifikke formål. I Norge har vi eksempelvis tjenester som [Meteorologisk institutt](#), [Norgeskart](#), [Geonorge](#), [Norge i bilder](#) og [Vegkart](#).

Disse webbaserte kart- og navigasjonsapplikasjoner bruker også geodata og GIS-teknologi for å presentere geografiske informasjon. De er generelt enklere å bruke og krever ikke spesialisert GIS-kompetanse, noe som gjør dem mer tilgjengelige for et bredt publikum. På den annen side krever GIS-systemer vanligvis profesjonell kompetanse og erfaring for å kunne utnytte deres fulle potensiale innenfor mer komplekse geografiske analyser og datahåndtering.

Vurderingsseksempler

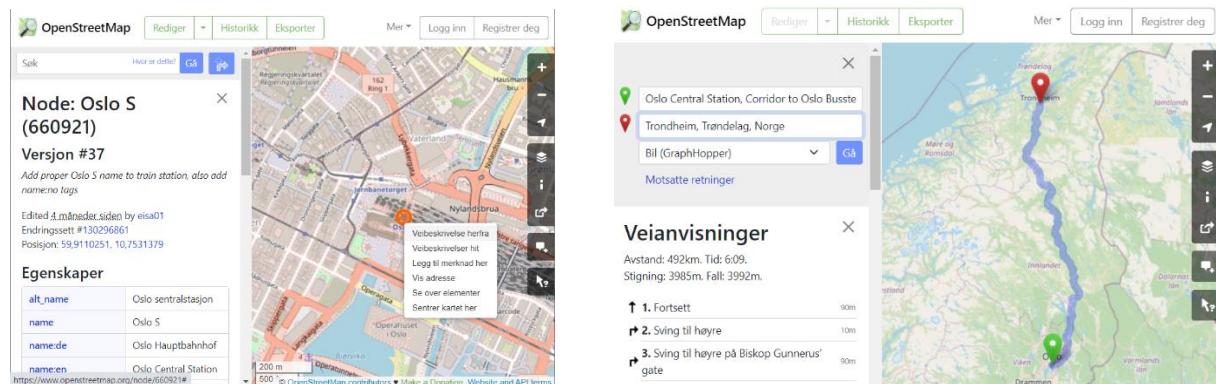
# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### OpenStreetMap

De fleste har hørt om eller brukt Google Maps, som gir brukere tilgang til veibeskrivelser, stedsinformasjon, satellittbilder og annen kartrelatert informasjon. Færre har imidlertid hørt om [OpenStreetMap \(OSM\)](https://www.openstreetmap.org/), et åpent og fritt tilgjengelig kartprosjekt som tillater brukere å bidra til og redigere kartdata, en slags "Wikipedia" for kart. Selv om det kan være vanskelig å konkurrere med Google Maps, er OSM et spennende prosjekt som kan være verdt å utforske. Hvis vi går til <https://www.openstreetmap.org/> og søker på "Oslo S", kan vi klikke på den første noden og få fram kartet som vist under til venstre.

Nå kan vi høyre-klikke på punktet i kartet for å finne avstanden og veianvisninger til et annet sted. Her ser vi at det er 49,2 mil til Trondheim. OSM lar oss også søke etter andre steder som skoler, sykehus og svømmehaller, selv om det ikke er sikkert at alle er lagt inn.



### OSMnx

[OSMnx](#) er en Python-pakke vi kan bruke til å hente, analysere og visualisere kartdata fra OpenStreetMap på en enkel måte. Det lar oss lage kart over veier, fotgjengerstier, sykkelstier og kollektivtransportruter, og utføre ulike operasjoner som å finne avstander, reisetider og generere ruter. Det er et populært verktøy blant forskere, planleggere og andre som jobber med kart og geografiske data for å studere byer, transport og geografiske mønstre. Det er viktig å merke seg at tilgjengeligheten av data i OSMnx kan variere avhengig av geografisk område. Mens det er mye detaljert data tilgjengelig for tettbebygd områder som byer, kan det være mindre detaljer tilgjengelig for mer landlige områder i Norge. I tillegg kan det ta tid å laste ned store mengder data for å utføre komplekse analyser eller finne ruter for store geografiske områder.

I eksempelet til høyre oppretter vi en [GeoDataFrame](#) for Oslo og bruker koordinatene til midtpunktet som er 59.9755 N og 10.7385 Ø. Vi gjør det tilsvarende for Trondheim og bruker OSMnx sin [distance.great\\_circle\\_vec](#)-metode til å finne avstanden mellom de to byene, som er 37.8 mil i luftlinje.

```
1 import osmnx as ox
2
3 # Skriv ut koordinatene til Oslo
4 oslo = ox.geocode_to_gdf("Oslo, Norway")
5 oslo = oslo["geometry"][0].centroid
6 print(f"Koordinatene til Oslo er {oslo.y:.4f} N, {oslo.x:.4f} Ø")
7
8 # Finn luftlinjeavstand mellom Oslo og Trondheim i mil
9 trondheim = ox.geocode_to_gdf("Trondheim, Norway")
10 trondheim = trondheim["geometry"][0].centroid
11 avstand = ox.distance.great_circle_vec(
12     oslo.y, oslo.x, trondheim.y, trondheim.x) / 10000
13 print(f"Avstanden mellom Oslo og Trondheim er {avstand:.1f} mil.")
14
15 0.1s
```

Koordinatene til Oslo er 59.9755 N, 10.7385 Ø  
Avstanden mellom Oslo og Trondheim er 37.8 mil.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

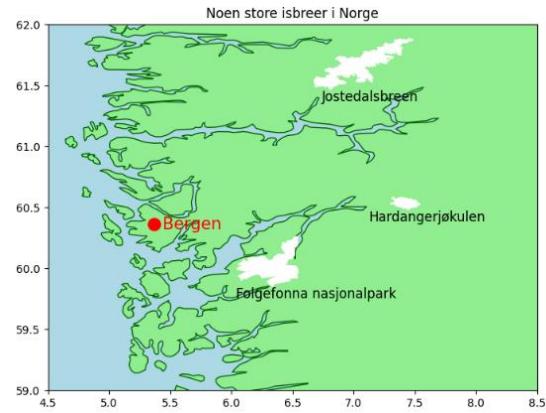
### OSMnx og GeoPandas (Isbreer)

Geodata som vi henter med OSMnx, kan vi vise i kart med GeoPandas og Matplotlib. Her har vi som tidligere plottet Norge fra verdenskartet i `countries.geojson`. Etter å ha hentet geodataene for isbreene Jostedalsbreen, Folgefonna og Hardangerjøkulen og lagret dem i en GeoDataFrame, bruker vi `plot`-metoden til å tegne geometrien på kartet. `plot`-metoden tar inn et `ax` (axes) argument, som angir området hvor det grafiske innhold blir tegnet. Vi kan også angi forskjellige egenskaper som farge, kantfarge, og mer for å tilpasse utseendet til plottet. Her brukes `plot`-metoden til å plotte isbreene som hvite polygoner med hvite kanter på kartet samt skrive navnet under.

```
# Hent isbreene fra OpenStreetMap
navn = ['Jostedalsbreen', 'Folgefonna', 'Hardangerjøkulen']
isbre = [ox.geocode_to_gdf(f'{isbre}', Norway) for isbre in navn]

# Plott isbreene
for isbre in isbre:
    isbre.plot(ax=ax, color='white', edgecolor='white')
    navn = isbre['display_name'][0].split(',')[0]
    # sett x til midtpunktet i polygonet
    x = isbre['geometry'][0].centroid.x-0.3
    # sett y til minste x-verdi i polygonet
    y = isbre['geometry'][0].bounds[1]-0.1
    plt.text(x, y, navn, fontsize=12, color='black')

# Vis Bergen som referanse
bergen = ox.geocode_to_gdf('Bergen, Norway')
sentrum = bergen['geometry'][0].centroid
y, x = sentrum.y, sentrum.x
ax.add_patch(Circle((x, y), 0.05, color='red'))
plt.text(x+0.07, y-0.04, 'Bergen', fontsize=15, color='red')
```



### Folium

[folium \(docs\)](#) er et Python-bibliotek som gjør det enkelt å lage interaktive Leaflet-kart som kan vises både i en nettleser og i Jupyter Notebook. Leaflet er et JavaScript-bibliotek og et av de mest populære verktøyene for å lage lage interaktive kart på nettsider og i mobilapplikasjoner. Vi kan bruke Folium sammen med OSMnx for å hente inn geodata fra OpenStreetMap og visualisere det på kartet. Leaflet-kart ser mer realistiske ut enn om vi hadde brukt GeoPandas. Generelt sett er Folium bedre egnet til å lage kart for visning, mens GeoPandas er bedre egnet til å analysere geodata.

Koden til høyre bruker `osmnx` og `folium` til å hente og visualisere informasjon om skoler i Indre Østfold, Norge. Vi begynner med å hente geometriene (koordinatene) til alle skoler i området ved hjelp av `osmnx`-biblioteket.

`tags={'amenity': 'school'}` sier til `geometries_from_place` at vi bare ønsker å hente ut geometrier som har en spesiell *tag*-kombinasjon som definerer skoler, `amenity=school`. Dette betyr at funksjonen bare henter ut geometrier som er klassifisert som skoler i OpenStreetMap-databasen. Det er mulig å søke etter ulike typer fasiliteter i OpenStreetMap, og en liste over disse finner vi på siden <https://wiki.openstreetmap.org/wiki/Key:amenity>.

Deretter finner vi sentrum av skolegeometriene og lager et `folium`-kart som sentrerer på dette punktet. Til slutt bruker vi en løkke for å legge til en markør for hver skole på kartet,

```
import osmnx as ox
import folium as fol

skoler = ox.geometries_from_place(
    'Indre Østfold, Norway', tags={'amenity': 'school'})
sentrum = skoler.unary_union.centroid
m = fol.Map(location=[sentrum.y, sentrum.x], zoom_start=11)

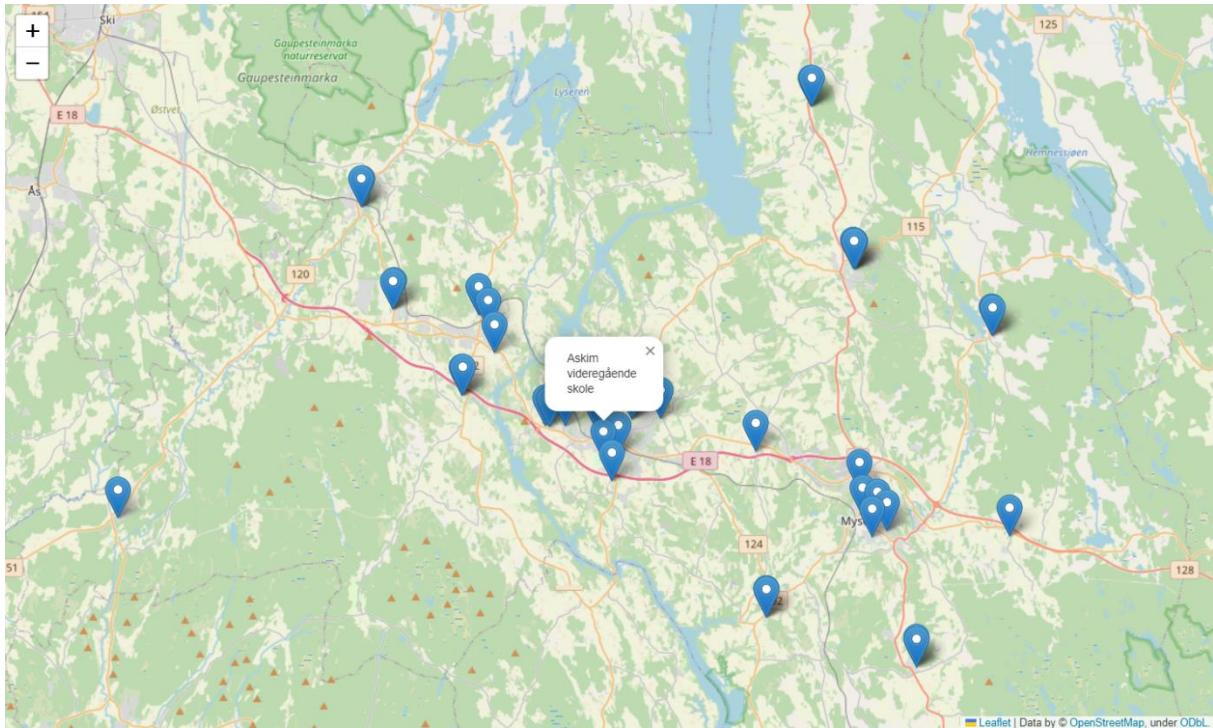
for geo, name in skoler[['geometry', 'name']].values:
    sentrum = geo.centroid
    fol.Marker((sentrum.y, sentrum.x), popup=name).add_to(m)

m.save('skoler.html')
```

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

med navnet på skolen som en **popup**-tekst. Til slutt lagres kartet som en HTML-fil ved navn **skoler.html**. **m** på det siste linjen viser kartet i Jupyter Notebook.



## Øvingsoppgaver

### 5.10.1

Bruk [Se eiendom](#) (kartverket.no) til å vise et kart av eiendommen der du bor. Klipp ut og lim inn bildet i besvarelsen.

### 5.10.2

Åpne [Google Maps](#), høyre-klikk på slottet og registrer bredde- og lengdegrad til Oslo. Sammenlign med koordinatene med de fra [geopy](#).

### 5.10.3

Åpne [OpenStreetMap](#). Finn veiavstanden mellom Oslo og Bergen. Sammenlign avstanden med den fra [geopy](#).

### 5.10.4

Åpne [Norge i Bilder](#) og vis et tredimensjonalt bilde av Vamma Kraftstasjon ved Askim

### 5.10.5

Utforsk filene [4\\_3\\_5\\_a.ipynb](#) og [4\\_3\\_5\\_b.ipynb](#). Begge programmene leser inn to filer, [kommunegrenser.geojson](#) ([kilde](#)) og [kommunebefolking.csv](#) ([kilde](#)). Programmene viser hver sin måte å flette inn befolkningsdata fra [kommunebefolking.csv](#) til en [GeoDataFrame](#) basert på [kommunegrenser.geojson](#). Deretter kan vi plotte et kart med farger basert på størrelsen på befolkningen. Sjekk også ut [Choosing Colormaps in Matplotlib](#).

Vurderingsseksemplar

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

#### 5.10.6

Lag et Folium-kart som viser minibanker innenfor 1 km radius fra slottet i Oslo. Du kan bruke `dist=1000` som argument i `features_from_point`.

#### 5.10.7

Denne oppgaven ble gitt på det første eksempelsettet til eksamen IT-2 som kom høsten 2022 og er senere fjernet. Det var en fortsettelse av Oppgave 14, Diagrammer over sykkelturer, som vi løste i 2.8 Reelle datasett med JSON. Bruk datasettet i 05.csv eller 05.json.

## Oppgave 15 - Plotting på kart

Du skal lage et program som leser inn informasjon fra datasettet og presenterer dette på et kart. Du skal bruke datasettet fra forberedelsen. Hvis du ikke har forberedt dette kan du også laste ned datasettet fra forberedelsesdelen nå.

Krav:

- Programmet skal generere et kart sentrert i Oslo sentrum.
- Kartet skal være tilpasset med en fornuftig zoom og det skal være mulig å zoome inn og ut i kartet.
- Programmet skal generere ikoner for hver startlokasjon og plassere disse på riktig posisjon i kartet.
- Størrelsen på ikonene skal samsvare med antall turer som startet på den aktuelle lokasjonen, for eksempel at radius er frekvens delt på 100 eller en annen passende verdi.

## 5.11 API for web og mobil med Flask og Dash

### Innhold

Introduksjon .....	365
SQLite3 .....	366
Flask .....	366
Dash .....	369
Publisering til Internett .....	372
DBeaver .....	374
Datasikkerhet .....	374

### Introduksjon

Python-programmene vi har laget hittil, har vært standard skrivebordsapplikasjoner som kan kjøres på en PC eller Mac. Vi startet med å bruke et kommandolinjegrensesnitt (**CLI**) i terminalvinduet, og gikk deretter over til et grafisk brukergrensesnitt i **tkinter**- eller **pygame**-vinduer. Vi har også vist diagrammer i **matplotlib**-vinduer, og eksportert disse diagrammene til **HTML**-sider og **PDF**-dokumenter ved å bruke **Jupyter**-notebooks. Til slutt inkluderte vi matplotlib-diagrammer i tkinter-vinduer. La oss nå se på hvordan vi kan flytte Python-programmene våre til skyen og gjøre dem tilgjengelige via en nettleser, slik at de kan nås og brukes av flere brukere uansett hvor de er.

I dagens digitale verden er mobiltelefoner en uunnværlig del av hverdagen for de fleste. Vi når derfor ut til langt flere brukere dersom vi lager programmene våre for mobile enheter. Igjen har vi flere valgmuligheter. Vi kan lage mobilapper, nettapplikasjoner eller hybride løsninger. Bibliotekene **Kivy** og **Beeware** lar oss utvikle mobilapper med Python. **Flask** og **Django** er rammeverk for å utvikle webapplikasjoner, hvor Flask er enklere og Django mer avansert. Vi kan også kjøre Python-programmer på mobile enheter ved hjelp av emulerte miljøer som **Termux** eller **QPython**. Hybride løsninger refererer ofte til apper som kjører i en app som simulerer en nettleser, og de blir vanligvis ikke utviklet med Python, men med verktøy som **React Native** og **Ionic**.

Mobilapper egner seg godt for brukere på farten eller for å utnytte enhetens funksjonalitet, som for eksempel kamera og GPS. Nettapper er derimot bedre egnet for å gi tilgang fra flere forskjellige enheter, inkludert både stasjonære datamaskiner og mobile enheter, samt å oppdatere applikasjonen uten at brukerne behøver å installere oppdateringer på enhetene sine.

Her skal vi se nærmere på Flask og Dash, to populære Python-rammeverk for å utvikle webapplikasjoner. Mange forbinder webutvikling med PHP på serversiden og JavaScript på klientsiden. Både Flask og Dash kjøres på serversiden, men vi skal bruke Flask som *backend* og Dash som *frontend*. *Backend* refererer til serversiden av en applikasjon, som håndterer

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

logikk, databaseinteraksjoner, brukergodkjenning, osv. *Frontend* er den delen av applikasjonen som håndterer brukergrensesnittet. Dash genererer *frontend*-innholdet på serversiden, men det sendes til og kjøres i brukernes nettlesere. Dette betyr blant annet at brukerne ikke får tilgang til koden på samme måte som med JavaScript.

### SQLite3

Vi planlegger å lage en webapplikasjon med data som kan opprettes, leses, redigeres og slettes. Python kommer med en SQL-database som er forholdsvis enkel å sette opp. For å komme i gang må vi importere `sqlite3` og opprette en forbindelse til databasen ved hjelp av `sqlite3.connect`, som vi gjør i metoden `get_db`. Hvis databasefilen ikke finnes, vil den bli opprettet automatisk.

I `__init__`-metoden kaller vi `init_db()`, som igjen kaller `get_db()` for å etablere en kobling til databasen. Deretter kalles `create_table()` for å opprette tabellen vi skal bruke, hvis den ikke allerede finnes. Andre deler av applikasjonen kan nå bruke `get_db()` for å utføre SQL-setninger på databasen.

`g` er et spesielt objekt som Flask bruker til å lagre data midlertidig for REST API-forespørsler. Det sørger dessuten for at databasetilkoblingen utnyttes effektivt og at den lukkes kontrollert ved hjelp av `teardown_appcontext`.

`Database`-klassen ligger i filen `flask_baskend.py` i mappen

`\eksempler\tilleggsstoff_5_11_API_for_web_og_mobil\flask`

```
import sqlite3
import flask
from pathlib import Path

class Database:
    def __init__(self, app: flask.Flask):
        self.db_path = Path(__file__).resolve(). \
            parent.joinpath('flask.sqlite')
        self.app = app
        self.init_db()
        self.app.teardown_appcontext(self.close_db)

    def init_db(self):
        with self.app.app_context():
            db = self.get_db()
            self.create_table(db)
            db.commit()

    def get_db(self):
        if 'db' not in flask.g:
            flask.g.db = sqlite3.connect(
                self.db_path,
                detect_types=sqlite3.PARSE_DECLTYPES
            )
            flask.g.db.row_factory = sqlite3.Row
        return flask.g.db

    def create_table(self, db):
        db.cursor().executescript("""
-- Opprett tabellen personer hvis den ikke eksisterer
CREATE TABLE IF NOT EXISTS personer (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    navn TEXT,
    alder INTEGER,
    bosted TEXT
);
""")

    def close_db(self, exception):
        db = flask.g.pop('db', None)
        if db is not None:
            db.close()


```

### Flask

Flask er et Python-rammeverk for å utvikle webapplikasjoner og REST API. Vi skal bruke Flask til å lage et REST API. Flask fokuserer først og fremst på serversiden av webapplikasjoner. Selv om vi kan lage websider med Flask, krever det ofte kunnskap om HTML, CSS, JavaScript og [Jinja2](#) for å lage fullverdige webapplikasjoner. Derfor vil vi heller bruke Dash, ettersom det



# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

bygger på Flask og ikke krever HTML/CSS/JavaScript-kompetanse. Vi finner mye dokumentasjon og mange veiledninger på [Flask](#)-nettstedet.

Vi installerer Flask med `pip install Flask`, lager et lite Python-program som heter `hilsen.py` og starter serveren med appen fra terminalvinduet med `flask --app hilsen run`. Resultatet kan vi se i en nettleser med URL-en `localhost:5000`. Se filen `hilsen.py`.

The image shows two side-by-side windows. On the left is a code editor with the file 'hilsen.py' containing the following Python code:

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def hilsen():
7      return "<h1>Hilsen Smidig IT-2.</h1>"
```

On the right is a screenshot of a web browser window titled 'localhost:5000' with the URL 'http://localhost:5000'. The page displays the text 'Hilsen Smidig IT-2.' in a large, bold, black font.

I Flask refererer en *route* til en URL-sti som angir hvilken funksjon som skal håndtere forespørsler som matcher den angitte URL-en. Flask-ruter er definert ved hjelp av *route-dekoratoren* i Flask. Vi kan tenke på dem som interne lenker på et webområde. På samme måte som når vi klikker på en lenke for å navigere til en annen side på et webområde, kan vi sende en forespørsel til en bestemt Flask-rute for å utføre en bestemt funksjon. I dette tilfellet returnerer funksjonen en webside med overskriften "Hilsen Smidig IT-2".

Som nevnt skal vi bruke Flask til å utvikle et REST API som kan opprette, lese, endre og slette data fra en database. Vi har valgt å lage vår egen `Server`-klass. Når en ny instans av denne klassen opprettes, utfører den følgende handlinger:

```
class Server:
    def __init__(self):
        self.flask = flask.Flask(__name__)
        self.database = Database(self.flask)
        self.setup_routes()
```

- Oppretter en `Flask`-server (en instans av `Flask`-klassen).
- Oppretter en instans av vår egen `Database`-klass, som kobler seg til `SQLite3`-databasen.
- Kjører vår egen `setup_routes`-metode, som definerer ruter med tilhørende funksjoner for tjenestene vi tilbyr..

Når vår `Server`-instans er opprettet, utfører vi `server.flask.run()`, som aktiverer `Flask`-serveren og gjør den tilgjengelig for å motta og håndtere forespørsler på standardporten 500.

```
if __name__ == "__main__":
    server = Server()
    server.flask.run(debug=True)
```

Vi kan hente data fra databasen direkte fra en nettleser ved hjelp av GET-metoden, som er standard når ingen metode er spesifisert i URL-en. For å sende POST-, PUT- og DELETE-forespørsler finnes det flere alternativer. Vi skal bruke `curl` (Client URL) fra kommandolinjen for å teste API-et.

```
> $PSVersionTable.PSVersion
Major Minor Patch PreReleaseLabel BuildLabel
---- ---- - - -
7       4       4

> curl --version
curl 8.7.1 (Windows) libcurl/8.7.1 Schannel zlib/1.3 WinINN
Release-Date: 2024-03-27
```

I Windows kan det være nødvendig å oppdatere til den nyeste versjonen av [PowerShell](#) fra [Microsoft Store](#). Sjekk versjonen ved å skrive `$PSVersionTable.PSVersion` og `curl --version` i terminalvinduet i VS Code.

Vi trenger å kjenne til tre argumenter i `curl`-kommandoen:

- `-X` angir metoden: GET (standard), PUT, POST eller DELETE.
- `-H` header som forteller om vi ønsker å sende eller motta data i JSON-format.
- `-d` dataene som skal overføres.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

- Til slutt skriver vi URL-en for nettadressen.

Vi kan bruke samme rute for alle REST API-metodene. Det er metoden som er angitt i dekoratøren `@self.flask.route` som bestemmer hvilken funksjon i `Server`-klassen som skal kjøres. Når vi sender den viste POST-forespørselen til serveren, kjøres funksjonen `legg_til_person`. Den henter dataene fra meldingen og legger dem inn i databasen med en SQL `INSERT INTO`-kommando. Etterpå returneres en statusmelding. Se filen `flask_backend.py`. Hele `curl`-kommandoen kan skrives på én linje, men her har vi brukt mellomrom, ` (accent grave) og Shift-Enter for å gjøre det mer oversiktlig.

```
> curl `<http://localhost:5000/personer` -X POST -H "Content-Type: application/json" -H "Accept: application/json" -d '{"navn": "Bjarne Bodal", "alder": 48, "bosted": "Bergen"}'`
```

```
@self.flask.route("/personer", methods=['POST'])
def legg_til_person():
    try:
        db = self.database.get_db()
        new_person = flask.request.json
        db.execute(
            'INSERT INTO personer (navn, alder, bosted) VALUES (?, ?, ?)',
            (new_person['navn'], new_person['alder'],
             new_person['bosted']))
        db.commit()
        return flask.jsonify({"status": "OK",
                             "melding": "Personen ble lagt til"}), 201
    except Exception as e:
        return flask.jsonify({"status": type(e).__name__,
                             "melding": str(e)}), 500
```

For å hente ut alle postene i databasen, holder det med `curl http://localhost:5000/personer` fordi GET er standard. `Les_personer()` kjøres, utfører SQL `SELECT`-kommandoen og returnerer postene i JSON-format.

```
> curl http://localhost:5000/personer
[{"id": 1, "navn": "Bjarne Bodal", "alder": 48, "bosted": "Bergen"}, {"id": 2, "navn": "Hilde Holme", "alder": 27, "bosted": "Halden"}]
```

```
@self.flask.route("/personer", methods=['GET'])
def les_personer():
    try:
        db = self.database.get_db()
        personer = db.execute('SELECT * FROM personer').fetchall()
        return flask.jsonify([dict(person) for person in personer])
    except Exception as e:
        return flask.jsonify({"status": type(e).__name__,
                             "melding": str(e)}), 500
```

For å slette en person, angir vi `id`-en i URL-teksten. Når vi sender denne forespørselen, kjøres funksjonen `slett_person()`, som utfører en SQL `DELETE`-kommando for å fjerne personen fra databasen og returnerer en statusmelding. Dekoratøren `@self.flask.route("/personer/<int:id>", methods=['DELETE'])` gjør tallet i URL-en tilgjengelig for funksjonen som et heltall.

```
> curl `<http://localhost:5000/personer/1` -X DELETE`
```

```
@self.flask.route("/personer/<int:id>", methods=['DELETE'])
def slett_person(id):
    try:
        db = self.database.get_db()
        rowcount = db.execute(
            'DELETE FROM personer WHERE id = ?', (id,)).rowcount
        db.commit()
        if rowcount == 0:
            return flask.jsonify({"status": "Feil",
                                 "melding": "Personen ble ikke funnet"}), 404
        return flask.jsonify({"status": "OK",
                             "melding": "Personen ble slettet"}), 200
    except Exception as e:
        return flask.jsonify({"status": type(e).__name__,
                             "melding": str(e)}), 500
```

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

For å redigere informasjonen om en person, bruker vi PUT. Når vi sender denne forespørselen, kjøres funksjonen `rediger_person()`, som utfører en SQL `UPDATE`-kommando for å oppdatere databasen, og returnerer en statusmelding. `id`-verdien i JSON-data brukes til å identifisere personen som skal endres. Hilde Holme har tatt navnet `Heen`, blitt `67` år og flyttet til `Hov` i Søndre Land.

```
> curl `<http://localhost:5000/personer` -X PUT -H "Content-Type: application/json" -H "Accept: application/json" -d '{"id":2, "navn": "Hilde Heen", "alder": 67, "bosted": "Hov"}'
> curl http://localhost:5000/personer
[{"melding": "Personen ble oppdatert", "status": "OK"}]
```

```
@self.flask.route("/personer", methods=['PUT'])
def rediger_person():
    try:
        db = self.database.get_db()
        redigert_person = flask.request.json
        id = redigert_person['id']
        rowcount = db.execute(
            'UPDATE personer SET navn = ?, \
            alder = ?, bosted = ? WHERE id = ?',
            (redigert_person['navn'], redigert_person['alder'],
             redigert_person['bosted'], id)
        ).rowcount
        db.commit()
        if rowcount == 0:
            return flask.jsonify({"status": "Feil",
                                  "melding": "Personen ble ikke funnet"}), 404
        return flask.jsonify({"status": "OK",
                              "melding": "Personen ble oppdatert"}), 200
    except Exception as e:
        return flask.jsonify({"status": type(e).__name__,
                              "melding": str(e)}), 500
```

## Dash

Nå som vi kan lage REST API-er med Flask, er det på tide å lage nettsidene som skal vise fram informasjonen. Det er her Dash kommer inn i bildet. Dash er et rammeverk i

Python som lar oss lage interaktive *dashboards* og datagrafikk på en enkel måte. Mens det kan være nyttig å kunne HTML og CSS for å lage mer skreddersydde løsninger, er det ikke nødvendig for å bruke Dash og lage nettsider. Dash har et enkelt og intuitivt grensesnitt som gjør det lett for utviklere å lage nettsider på en oversiktlig og brukervennlig måte.

For å komme i gang med Dash, trenger vi bare å installere det med `pip install dash`, skrive litt Python kode i en fil som vi kaller `hilsen.py` s om ligger i mappen

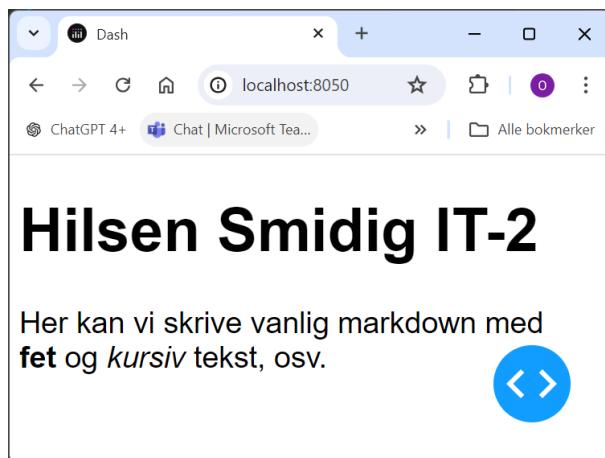
`\eksempler\tilleggsstoff_5_11_API_for_web_og_mobil\dash`. Deretter starter vi serveren ved å kjøre `hilsen.py` som et vanlig Python-program. Når serveren er startet, kan vi se resultatet i en nettleser ved å gå til URL-adressen `localhost:8050`.

```
import dash
import dash_core_components as dcc

app = dash.Dash(__name__)

app.layout = dcc.Markdown('''
# Hilsen Smidig IT-2
Her kan vi skrive vanlig markdown med
**fet** og *kursiv* tekst, osv.
''')

if __name__ == '__main__':
    app.run_server(debug=True)
```



`app` er en instans av Dash-klassen som er ansvarlig for å koble sammen alle delene av en applikasjon, inkludert layouten, tilkoblede data og interaktive funksjoner. `layout`-egenskapen

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

definerer utseendet og strukturen til websiden. Her bruker vi `dcc.Markdown`, som er en komponent i `dcc`-pakken (*Dash Core Components*). Ved å bruke *markdown*, som ble gjennomgått i forbindelse med [Jupyter i bok 3.1](#), kan vi enkelt formaterere tekst uten å måtte kunne HTML. Det finnes mange veiledninger for Dash på <https://dash.plotly.com/>.

Dash kjører også på en Flask-server, så vi kan forenkle ved å samle *backend* og *frontend* i samme modul, på samme server, og til og med på samme port. Derfor inkluderer vi både `Database`- og `Server`-klassene i `dash_frontend.py`. Når vi kjører dette programmet, betjener ruten / Dash-delen, og /personer betjenes av Flask. Etter at vi har opprettet en `Server`-instans med en Flask-server, bruker vi denne når vi oppretter Dash-instansen.

```
if __name__ == "__main__":
    server = Server()
    dash_app = DashApp(server.flask)
    server.flask.run(debug=True)
```

Når vi oppretter en instans av `DashApp`-klassen, lages først en `Dash`-instans. `self.data_server` vil inneholde data slik de er lagret i databasen. Når brukeren endrer dataene i tabellen på nettsiden, sammenligner vi disse med `self.data_server` for å identifisere hvilke oppdateringer som må sendes til serveren for lagring i databasen. I `self.setup_layout` utformer vi websidens utseende og i `self.setup_callbacks` definerer vi handlingene som skal utføres når ulike hendelser inntreffer, for eksempel endringer i tabellen.

```
class DashApp:
    def __init__(self, flask_app: flask.Flask) -> None:
        self.app = dash.Dash(__name__, server=flask_app,
                             routes_pathname_prefix='/')
        self.app.title = 'DashApp'
        self.data_server = []
        self.setup_layout()
        self.setup_callbacks()
```

I `layout` pakker vi alt sammen inn i en `dash.html.Div`-komponent. `dash.dcc.Interval` brukes for å gjenta forespørsler om å hente data fra databasen, dersom den ikke er klar ved oppstart. `dash.dcc.Tabs` er faner, og `dash.dcc.Tab` er den første fanen, `Hjem`, med en statisk webside skrevet i *Markdown*.

```
def setup_layout(self) -> None:
    self.app.layout = dash.html.Div([
        dash.dcc.Interval(id='interval_id',
                           interval=3000, disabled=False),
        dash.dcc.Tabs(id='tabs', value='home', children=[
            dash.dcc.Tab(label='Hjem', value='home', children=[dash.dcc.Markdown('''
# Hilsen Smidig IT-2
Her kan vi skrive vanlig markdown med
**fet** og *kursiv* tekst, osv.
''', style={'textAlign': 'center'})])]
```



The screenshot shows the same web browser window with the 'Persondata' tab selected. The content area displays the text 'Endringer lagret' above a table. The table has columns: 'Legg til rad', 'Id', 'Navn', 'Alder', and 'Bosted'. It contains two rows of data: row 1 with '1' and 'Bjarne Bodal', and row 2 with '2' and 'Hilde Heen'. At the bottom left of the table is a button labeled 'Legg til rad'.

Legg til rad	Id	Navn	Alder	Bosted
	1	Bjarne Bodal	48	Bergen
	2	Hilde Heen	67	Hov

```
dash.dcc.Tab(label='Persondata', value='persondata', children=[dash.dcc.Markdown(
    id='info_id', children='Henter data fra server...'),
    dash.html.Button(
        'Legg til rad', id='ny_rad_id', n_clicks=0),
    dash.html.Button('Lagre endringer',
                     id='lagre_id', n_clicks=0),
    dash.dash_table.DataTable(
        id='tabell_id',
        columns=[{'name': 'Id', 'id': 'id',
                  'type': 'numeric', 'editable': False},
                 {'name': 'Navn', 'id': 'navn',
                  'type': 'text'},
                 {'name': 'Alder', 'id': 'alder',
                  'type': 'numeric'},
                 {'name': 'Bosted', 'id': 'bosted',
                  'type': 'text'}],
        editable=True,
        sort_action='native',
        sort_by=[{'column_id': 'id', 'direction': 'asc'}],
        row_deletable=True,
        style_header={'fontWeight': 'bold',
                      'textAlign': 'center'})])
```

Den andre fanen, `Persondata`, starter med en *Markdown*-komponent for tilbakemeldinger til brukeren. Deretter følger to knapper med `dash.dcc.Button`. Den første brukes til å opprette en ny rad i tabellen, og den andre til å lagre endringer i databasen. Tabellen er en

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

`dash.dash_table.DataTable`. Vi tillater brukeren å redigere tabellen (`editable=True`), men ikke kolonnen `Id` (`editable=False`). Dessuten sikrer tabellen at det kun kan skrives inn sifre i `Alder`-feltene med `'type': 'numeric'`. `sort_action='native'` lar brukeren sortere tabellen ved å klikke i overskriftsraden, og `sort_by` er sorteringen ved oppstart. `row_deletable=True` legger til en ekstra kolonne til venstre med en `X` i hver rad, som kan klikkes på for å slette raden.

*Callback-funksjonene angis med `callback`-dekoratøren. Vi har kun én kombinert `callback`-funksjon fordi flere funksjoner ikke kan oppdatere samme komponent som angis i `Output` med komponent-ID og hvilken egenskap i denne komponenten som skal oppdateres. `Input` definerer hvilke komponenter og egenskaper vi skal lytte på. Når disse endres, kalles `callback`-funksjonen. `n_intervals`, `ny_rad_n_clicks`, og `lagre_n_clicks` inneholder hvor mange ganger *timeren* og knappene har blitt aktivert, men vi har ikke behov for å bruke disse. Når vi bruker samme funksjon til flere input, lar `dash.callback_context` oss finne ut hvilken komponent og egenskap som trigget hendelsen.*

```
def setup_callbacks(self) -> None:
    @self.app.callback(
        dash.Output('tabell_id', 'data'),
        dash.Output('interval_id', 'disabled'),
        dash.Output('info_id', 'children'),
        dash.Input('interval_id', 'n_intervals'),
        dash.Input('ny_rad_id', 'n_clicks'),
        dash.Input('tabell_id', 'data'),
    )
    def kombinert_callback(n_intervals: int, ny_rad_n_clicks: int,
                           lagre_n_clicks: int, data: list) -> tuple:
        ctx = dash.callback_context
        trigger_id = ctx.triggered[0]['prop_id'].split('.')[0]
        if trigger_id == 'interval_id':
```

Hvis det er *timeren* (`Interval`) som har utløst hendelsen, kalles `hent_data()`. Den foretar en GET-forespørsel til Flask-serveren, og dersom alt går greit, returneres en liste med ordbøker fra JSON-data. Hvis ikke, fanges dette opp av unntakshåndteringen, og `hent_data()` returnerer `None`. I så fall sendes ingenting (`dash.no_update`) tilbake til tabellen og *timeren*, men `Markdown`-komponenten `info_id` får teksten 'Kunne ikke hente data. Prøver igjen om 3 sekunder'. Dersom forespørselen var vellykket, settes `self.data_server` til listen som ble hentet. Deretter sendes denne listen til tabellens `data` for fremvisning. Samtidig settes *timerens* `disabled` til `True`, så den kobler ut, og `info_id` får teksten 'Hentet data'.

```
if trigger_id == 'interval_id':
    data_hentet = hent_data()
    if data_hentet is None:
        return dash.no_update, dash.no_update, \
               "Kunne ikke hente data. Prøver igjen om 3 sekunder"
    else:
        self.data_server = data_hentet
        return data_hentet, True, "Hentet data."
```

Hvis brukeren klikker på knappen for å legge til en rad, legges en tom rad til tabellens data som kom med `Input()` og sendes tilbake med `Output()`. Tabellen viser nå en ny rad som ikke finnes i databasen, og derfor returneres samtidig teksten 'Ulagrede endringer' til `children`-attributtet i `Markdown`-komponenten som viser meldingen.

```
elif trigger_id == 'ny_rad_id':
    data.append({'id': None, 'navn': None,
                 'alder': None, 'bosted': None})
    return data, dash.no_update, "Ulagrede endringer"
```

Hvis tabellen er endret, sammenlignes den med innholdet i databasen som ligger lagret i `self.data_server`. Hvis de er forskjellige, returneres 'Ulagrede endringer' til `info_id`-komponenten. Hvis ikke, returneres 'Tabelldata lik serverdata'. Det går jo an å legge til en rad, for så å slette den igjen.

```
elif trigger_id == 'tabell_id':
    if data != self.data_server:
        return dash.no_update, dash.no_update, \
               "Ulagrede endringer"
    else:
        return dash.no_update, dash.no_update, \
               "Tabelldata lik serverdata"
```

Nå gjenstår den mest omfattende delen av programmeringen. Hvis tabellen er endret, har brukeren i prinsippet to valg: Ved å laste inn websiden på nytt går endringene tapt, og dataene fra databasen vises på nytt. Men hvis brukeren klikker på

Vurderingsseksemplar

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

"Lagre endringer"-knappen, vil databasen bli oppdatert med dataene som vises i dash-tabellen. Vi sjekker først om de er like. Hvis de er det, er det ingen endringer å lagre, og vi kan returnere med denne teksten. Hvis de er forskjellige, kan vi ikke uten videre slette alle radene i databasen og legge inn alle radene fra tabellen på websiden. Hvorfor? I stedet finner vi ut hvilke rader som er nye, endret eller slettet. For å effektivisere prosessen lager vi en ordbok, `data_server_oppslug`, som bruker ID som nøkkel (`key`) og raden som verdi (`value`).

```
elif trigger_id == 'lagre_id':
    if data == self.data_server:
        return dash.no_update, dash.no_update, \
               "Ingen endringer å lagre"
    else:
        data_server_oppslug = {
            item['id']: item
            for item in self.data_server}
```

Nye rader er de som ikke har fått noen ID ennå.

Den tildeles når vi oppretter raden. Husk:

`id INTEGER PRIMARY KEY AUTOINCREMENT`. Listen med nye rader sendes som argument til `legg_til_personer()`, som returnerer en feilmelding hvis noe gikk galt med forespørslene. I `legg_til_personer()` utføres en POST-forespørsel for hver nye rad.

```
personer_nye = [item for item in data if not item['id']]
feil = legg_til_personer(personer_nye)
if feil:
    return dash.no_update, dash.no_update, feil
```

Endrede rader er de som har samme ID, men ulikt innhold. På samme måte som ovenfor sendes disse radene til `endre_personer()`, som utfører PUT-forespørsler.

```
personer_endret = [item for item in data
                    if item['id'] in data_server_oppslug
                    and item != data_server_oppslug[item['id']]]
feil = endre_personer(personer_endret)
if feil:
    return dash.no_update, dash.no_update, feil
```

Slettede rader er de med ID som finnes i `self.data_server`, men ikke i `data`. Vi har valgt å bruke differensmengden mellom to sett, se [3.3 Sett og boolsk algebra](#). Settet sendes som argument til `slett_personer`, hvor det utføres DELETE-forespørsler for hver ID.

```
# Lag sett med ID-er til slettede personer
# Ikke ta med nye rader uten ID
data_id_sett = {d['id']
                 for d in data if d['id'] is not None}
data_server_id_sett = {item['id']
                         for item in self.data_server}
id_sett_slettet = data_server_id_sett - data_id_sett
feil = slett_personer(id_sett_slettet)
if feil:
    return dash.no_update, dash.no_update, feil
```

Hvis alt har gått bra så langt, kallas `hent_data()` for å verifisere at dataene fra databasen nå er like de dataene vi lagret.

```
data = hent_data()
if data is dash.no_update:
    return dash.no_update, False, \
           "Kunne ikke hente data. Prøver igjen om 3 sekunder"
else:
    self.data_server = data
    return data, True, "Endringer lagret"
```

Dersom `trigger_id` ikke skulle samsvare med noen av verdiene vi har sjekket, returnerer vi `dash.no_update` til alle tre Output: `data` til `DataTable`, `disable` til `Interval` og `children` til `Markdown`.

### Publisering til Internett

For at andre skal kunne se og bruke appen vår, må vi publisere den på internett. De som allerede har sin egen webplattform, anbefales å kjøre Flask via [Waitress](#) eller [Gunicorn](#). Disse er WSGI-servere (*Web Server Gateway Interface*), spesielt designet for effektiv håndtering av forespørsler i produksjonsmiljøer. Vi skal derimot bruke **Vercel**, som enn så lenge (2024) tilbyr gratis hosting for private prosjekter og støtter databasetilkoblinger. Vercel støtter ikke SQLite3, men **PostgreSQL**, så vi må gjøre noen endringer i programmet vårt, men siden Vercel håndterer all serverlogikk og skalerer automatisk etter behov, er det ikke nødvendig å bruke en tradisjonell WSGI-server som Waitress eller Gunicorn.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

1. Ta utgangspunkt i mappen `/vercel` med filene `dash_vercel.py`, `requirements.txt`, `vercel.json` og undermappen `assets`. Filen `requirements.txt` er laget med `pipreqs`. som forklart i [6.2 Før eksamen](#).
2. Om nødvendig, last ned **Node.js** fra <https://nodejs.org/en> og installer programmet.
3. Verifiser installasjonen med kommandoene `node -v` og `npm -v` i terminalvinduet.
4. Installer **Vercel CLI** med kommandoen `npm install -g vercel` i terminalvinduet.
5. Kjør `vercel` i terminalvinduet for å opprette en konto og distribuere appen i utviklingsmodus. For å publisere appen i produksjonsmodus, kjør `vercel --prod`. Følg instruksjonene.

Produksjonslenken vil feile i første omgang, men *Inspect*-lenken leder til *Deployments*-siden hvor en PostgreSQL-database kan opprettes under *Storage*. Koble denne til prosjektet og sørge for at miljøvariablene (de som starter med POSTGRES\_) er opprettet under *Settings*. Under *Domains* finnes URL-adressen som skal brukes for Flask-serveren. Opprett en miljøvariabel FLASK\_URL for denne. Husk å ha med <https://> foran og ingen / til slutt..

The screenshot shows the Vercel Settings page for a project. On the left, there's a sidebar with links: Project, Deployments, Analytics, Speed Insights, Logs, Firewall, Storage, and Settings (which is selected). The main area lists environment variables:

Variable	Type	Value
FLASK_URL	Development, Preview, Production	<a href="https://dash-two-corral.vercel.app">https://dash-two-corral.vercel.app</a>
POSTGRES_DATABASE	Development, Preview, Production	verceldb
POSTGRES_HOST	Development, Preview, Production	ep-billowing-mountain-a4ktk03p-po...
POSTGRES_PASSWORD	Development, Preview, Production	.....
POSTGRES_USER	Development, Preview, Production	default

For å overføre applikasjonen fra lokal SQLite til PostgreSQL på Vercel, må vi gjøre noen endringer i programmet. Først må vi bruke miljøvariablene POSTGRES\_URL og FLASK\_URL for å konfigurere PostgreSQL-tilkoblingen og Flask-URLen. SQLite-databasetilkoblingen, som bruker en lokal filbasert database, må byttes ut med `psycopg2`-tilkoblingen for PostgreSQL. Husk å installere `psycopg2`-binary med `pip install psycopg2-binary`. I SQL-spørringene må plassholderne endres fra ? til %s, som er standard for `psycopg2`. Videre må opprettelsen av databasen endres fra SQLite-syntaks til PostgreSQL-syntaks, som eksempelvis `SERIAL` for autoincrement av ID-feltet. I `DashApp` må vi erstatte maskinens loopback-adresse `127.0.0.1` med FLASK\_URL-miljøvariabelen, som brukes for GET-, POST-, PUT- og DELETE-forespørslene. Vi brukte `127.0.0.1` i stedet for `localhost` fordi forespørslene kan utføres raskere dersom DNS-oppslaget skulle ta ekstra tid. Endringer er merket med kommentaren `# Endret for PostgreSQL`.

Denne appen er publisert på <https://dash-two-corral.vercel.app/>

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

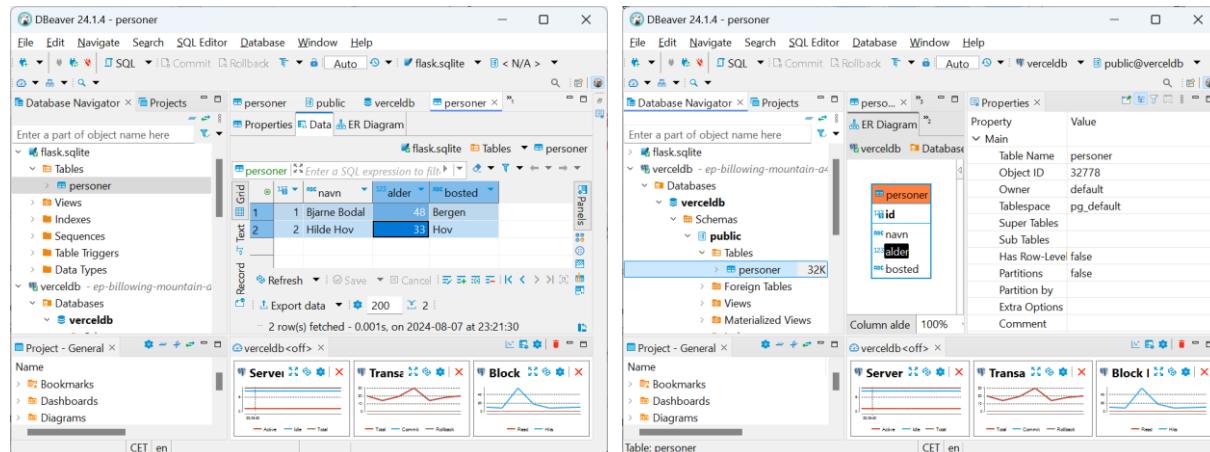
### DBeaver

Dersom vi ønsker å undersøke innholdet i databaser, har vi flere verktøy å velge mellom. Vi har valgt DBeaver, som er gratis og åpen kildekode. Programmet kan lastes ned fra <https://dbeaver.io/download/>, og det støtter mange forskjellige databaser, blant annet SQLite og PostgreSQL. Med sitt grafiske brukergrensesnitt gjør DBeaver det enkelt å administrere databaser, slik at vi kan inspirere data, kjøre SQL-spørninger og feilsøke problemer på en visuell og brukervennlig måte.

Før vi kan bruke DBeaver, må vi koble oss til databasen. For **SQLite**-databasen på vår maskin trenger vi bare å åpne filen, som vi har kalt **flask.sqlite**. For å koble oss til **PostgreSQL**-databasen på internett bruker vi **innholdet i miljøvariablene** som er oppgitt under *Settings -> Environment Variables* på prosjektsiden i Vercel:

- Host: POSTGRES\_HOST
- Database: POSTGRES\_DATABASE
- Username: POSTGRES\_USER
- Password: POSTGRES\_PASSWORD

På bildet under ser vi **dataene i SQLite-databasen** til venstre. Bildet til høyre viser egenskapene til **alder** i **PostgreSQL**-databasen. Vi er koblet opp mot begge databasene samtidig.



### Datasikkerhet

Når vi utvikler webapplikasjoner, er det viktig å tenke på sikkerhet for å beskytte både serveren og dataene mot potensielle angrep. For å holde koden enkel, har vi ikke inkludert disse tiltakene her. For websider i daglig bruk bør vi validere inndata, bruke innlogging (autentisering), bruke HTTPS for kryptering, begrense antall forespørsler innen et gitt tidsrom, logge og overvåke aktivitet, samt andre tiltak som brannmurer, sikkerhetskopiering, osv.

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## Vedlegg 6

### 6.1 Verktøy og metoder for systemutvikling

Systemutvikling og samarbeid			
Verktøy/Metode	Funksjon	Fordeler	Ulemper
<b>Pseudokode</b> Tekstbehandling	Beskrivelse av algoritme ved bruk av naturlig språk	Enklere å forstå og mindre detaljert	Kan være upresist og misvisende
<b>Flytskjema</b> draw.io som VS Code-utvidelse	Grafisk fremstilling av en algoritme	gir en rask og enkel oversikt over algoritmen i et dataprogram	tidkrevende og tungvint å vedlikeholde, begrenset modelleringsmuligheter
<b>Wireframes</b> Pencil	Visuell planlegging av layout og funksjonalitet	Reduserer kostnader, løser designproblemer og bedrer brukeropplevelsen	Kan begrense kreativitet, forsinke prosessen og fokusere for mye på detaljer
<b>UML diagrammer</b> PlantUML	Dokumenterer, visualiserer og kommuniserer systemdesign	Standardisert, forenkler samarbeid, reduserer feil og øker produktiviteten	Kan bli tidkrevende, komplekst og vanskelig å lære og forstå
<b>Live Share</b> VS Code-utvidelse	Samhandlingsverktøy for teamkommunikasjon og prosjektadministrasjon	Enkelt å samarbeide om kode i sanntid, kan øke produktiviteten og redusere kostnader	Krever VS Code kunnskaper, internettforbindelse, og kan være sårbart for sikkerhetsrisiko.
<b>Slack</b>	Samhandlingsverktøy for teamkommunikasjon og prosjektadministrasjon	Enkelt å kommunisere, dele filer, integreres med andre verktøy som GitLab og Trello	Kan føre til informasjonsoverbelastning og begrenset funksjonalitet i gratisversjon
<b>Trello</b>	Prosjektstyringsverktøy med visuelle kort og lister	Enkel visuell oversikt, samarbeidsfunksjoner og oppgavekart som kan tilpasses egne behov.	Begrenset funksjonalitet i gratisversjon, kan bli rotete med mange oppgaver
<b>Git</b> VS Code med Git, GitLab og GitHub	Versjonskontrollsysten som gir mulighet for samarbeid og kodedeling	Sporing av endringer, enkelt å samarbeide med andre utviklere, innebygd i VS Code	Litt bratt læringskurve, noe begrenset funksjonalitet i VS Code
<b>Jupyter Notebooks</b> som VS Code-utvidelse	Interaktive, nettbaserte notatbøker som kan deles.	kombinerer tekst, grafikk og kjørbar kode i ett og samme dokument, fint for eksperimentering, enkelt grensesnitt, VS Code-utvidelse,	Kan bli komplekst for større prosjekter og krevende beregninger, bør ikke brukes som erstatning for IDE, versjonskontroll og automatiske tester.
<b>Dokumentasjon</b> autodocstring og better comments som VS Code-utvidelser og pdoc	Verktøy for å dokumentere kode og API	Letter forståelse og samarbeid på tvers av prosjekter, automatisert API-dokumentasjon	Krever at man bruker tid på å lage og vedlikeholde dokumentasjon

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 6.2 Før eksamen

#### Innhold

Vurderingskriterier .....	376
Nettverk ressurser .....	376
DevDocs .....	377
requirements.txt.....	377
Mal .....	378

#### Vurderingskriterier

UDIR utgir eksempeloppgaver, tidligere eksamensoppgaver og vurderingskriterier på nettet. For REA3049, se [nettssiden](#). Vurderingskriteriene finnes i [eksamensveiledningene](#), som oppdateres årlig og kan lastes ned som PDF-filer. Se også filene i mappen [\eksempler\vedlegg\\_6\\_02\\_foer\\_eksamen](#). Vurderingskriteriene er inndelt i de fire kompetanseområdene samfunn og systemer, problemløsning, modellering og representasjon samt implementering. De beskrives på tre nivåer for karakterene 2, 4 og 6. Karakterene 1, 3 og 5 beskrives ikke. Oppgavesettet vil dessuten bestå av oppgaver fra fire ulike kategorier, som også er beskrevet i veilederingen.

#### Nettverk ressurser

Det finnes [nasjonale føringer for nettbaserte hjelpebidrifter til eksamen](#) i videregående skole. Fylkeskommunale skoleeiere har utarbeidet en felles oversikt over hvilke nettbaserte hjelpebidrifter som skal være tilgjengelig til sentralt gitt eksamen. Den skal justeres årlig. Udir henviser til Novari IKS, som er et interkommunalt selskap som ivaretar forvaltning og videreutvikling av fylkeskommunenes felles IT-systemer innen videregående opplæring, samferdsel og teknologi. Se [Til deg som skal ta eksamen](#). Denne [nettressurslisten](#) (.xlsx) viser nettbaserte hjelpebidrifter til eksamen 2024. På listen finnes for eksempel

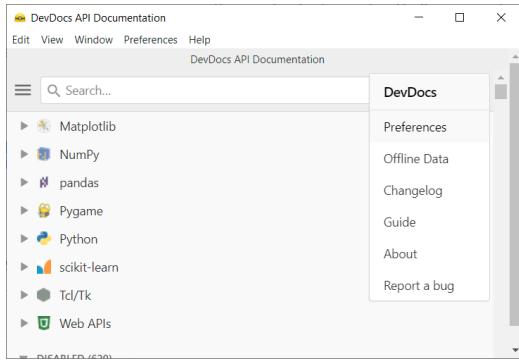
- [Lovdata](#), hvor vi blant annet kan finne hvilke [filformater](#) som er tillatt i offentlige arkiver.
- [NDLA](#) som eksempelvis har sider om [etikk](#) og [datasikkerhet](#).
- [W3Schools](#) hvor det er mange sider om [Python](#).

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

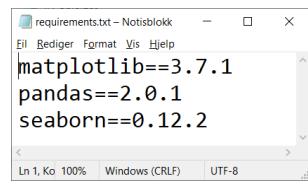
### DevDocs

DevDocs er et praktisk verktøy for programvareutviklere som gir tilgang til dokumentasjon for forskjellige programmeringsspråk, rammeverk og biblioteker. Vanligvis er DevDocs en online ressurs som er tilgjengelig gjennom internett, se <https://devdocs.io/>, men vi kan også laste ned dokumentasjonen på forhånd for å bruke den uten internettforbindelse under prøver eller eksamener når nettet er sperret. Desktop-programmet for Windows kan lastes ned fra <https://github.com/egoist/devdocs-desktop/releases>. Installer programmet, klikk på de tre prikkene til høyre i vinduet, velg *Preferences* og huk av de bibliotekene du vil benytte. Gjør det samme en gang til, men velg *Offline Data* denne gangen. Filene lagres i den skjulte **AppData**-mappen. DevDocs er enkelt og intuitivt å bruke, og som vi ser av nedtrekksmenyen, inneholder programmet også en brukerveiledning.



### requirements.txt

Når vi publiserer et Python-program, er det vanlig å inkludere en fil kalt **requirements.txt**. Denne filen inneholder informasjon om hvilke biblioteker og versjoner som er nødvendige for å kjøre programmet. Bibliotekene kan installeres av deg som skal bruke programmet, med kommandoen **pip install -r requirements.txt**. For å lage denne må vi først installere pipreqs: **pip install pipreqs**. I stedet for **pip freeze**, som lister ut alle bibliotekene som er installert globalt (hvis vi ikke er i et virtuelt miljø), bruker vi **pipreqs**. (mellomrom og punktum) i terminalvinduet i mappen hvor programmet/modulen/.py-filen ligger. Da får vi med bare de bibliotekene programmet trenger. Biblioteker som blir importert i programmet og er med i standard biblioteket i Python, blir ikke (og trengs ikke) tas med i **requirements.txt**. Informasjon om hvilken Python-versjon som kreves, bør vanligvis oppgis i README- eller setup.py-filene.



Filene går fort å lage. Python versjonen finner vi som kjent med **python --version**. Det anbefales å levere eksamsbesvarelser med **requirements.txt** og informasjon om hvilken Python-versjon som er benyttet.

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### Mal

Det kan være lurt å ha laget en mal for besvarelsen før selve eksamensdagen, så du slipper å bruke tid på å skrive inn hvilke programmer med tilhørende versjonsnumre som du har brukt. Se [forslag\\_til\\_mal\\_for\\_digital\\_eksamensbesvarelse.dotx](#). På eksamensdagen bør du lagre det endelige dokumentet som PDF.

Kandidatnummer		Eksamensdato	
Fagkode	REA3049	Fagnavn	Informasjonsteknologi 2

### Generelt

Jeg har benyttet læreverket *Smidig IT-2*, utgitt av TIP AS, skrevet av Ola Lie. Programmene er utviklet i Python ved bruk av VS Code, sammen med flere utvidelser. En detaljert oversikt over programmene og versjonene jeg har benyttet, finnes i vedlegget **Programmer og versjoner**.

I oppgaver med Python-kode, har jeg lagt ved **requirements.txt**. Denne filen gir en oversikt over alle nødvendige biblioteker, sammen med deres versjonsnummer, som ikke er en del av Python sitt standardbibliotek. Bibliotekene kan installeres med **pip install -r requirements.txt**.

Besvarelsene til de ulike oppgavene ligger i separate mapper, navngitt etter oppgavenummer, i tråd med angitte retningslinjer.

### Oppgave x

(Her er det mulighet for å inkludere utfyllende kommentarer, antagelser, tolkninger og mer, relatert til de ulike oppgavene, inkludert nettbaserte. Husk å kommentere ut eller fjerne kode som ikke fungerer. Det er viktig å ikke levere programmer som ikke kan startes eller kjøres. Mindre kjøretidsfeil kan forekomme, men bør unngås.)

### Vedlegg: Programmer og versjoner

(Slett de du ikke bruker)

- Word for Microsoft 365 MSO (Version 2304 Build 16.0.16327.20200) 64-biters)
- Python 3.10.8

**Smidig IT-2**  
**for eksklusiv bruk ved <navn på skole>**

## 6.3 Forankring i læreplanen

### Innhold

Innledning .....	379
Kompetansemål .....	379
Grunnleggende ferdigheter.....	379
Tverrfaglig tema.....	380
Kjerneelementer.....	380
Overordnet del .....	381

### Innledning

Skoleåret 2022/2023 begynte vi med [ny læreplan i Informasjonsteknologi 2](#) (LK20). De nye kompetansemålene i Informasjonsteknologi 2 representerer en betydelig endring i forhold til de gamle kompetansemålene (LK06). Mens den gamle læreplanen hadde et mer begrenset fokus på planlegging, dokumentasjon og teknisk utvikling av IT-løsninger, krever den nye læreplanen en mer helhetlig tilnærming til faget. Den nye læreplanen legger vekt på utforskning og vurdering av informasjonsteknologiens muligheter, utfordringer og konsekvenser i ulike sammenhenger, inkludert drøfting av etiske dilemmaer. En annen endring er innføringen av objektorientert modellering og programmering i læreplanen, og elevene skal nå lære å anvende konsepter som klasser, objekter, metoder og arv. Dette representerer et skifte fra den gamle læreplanen, som primært fokuserte på programmering med variabler, valg og gjentakelser, samt utvikling av multimedieapplikasjoner.

Gjenbrukbarhet av programkode, strategier for feilsøking og testing, vurdering av brukervennlighet i programmer og samarbeid med relevante systemutviklingsmetoder og -verktøy er også viktige elementer i den nye læreplanen, som i tillegg inkluderer kunnskap om standarder for datahåndtering og bruk av programmering for å innhente, analysere og presentere informasjon fra reelle datasett. Samlet sett representerer de nye kompetansemålene en bredere og mer helhetlig tilnærming til faget, som omfavner etikk, objektorientering, gjenbrukbarhet, brukervennlighet, samarbeid og moderne datahåndteringsteknikker.

### Kompetansemål

På slutten av hver runde har vi med en oppsummering hvor det vi har gjennomgått, er koblet til de ulike kompetansemålene. Dette forsikrer oss om at læreverket dekker samtlige kompetansemål, samtidig som det viser forankringen i læreplanen. Hver bolk innledes dessuten med de viktigste kompetansemålene som bolken omhandler.

### Grunnleggende ferdigheter

Vi har lagt vekt på å ivareta alle de grunnleggende ferdighetene i dette læreverket. Det er imidlertid viktig å merke seg at læreverket alene ikke fullt ut dekker de muntlige

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

ferdighetene. Det er opp til lærerne å legge til rette for diskusjoner i timen og gi elevene mulighet til å presentere og kommunisere sitt arbeid. Øvingsoppgavene som følger med læreverket, gir en god plattform for å styrke både den muntlige kommunikasjonen og skriveferdighetene. Leseferdighetene utvikles gjennom lesing av dette dokumentet, henvisninger til annen dokumentasjon og analyse av programkode. Videre vil elevene bli utfordret til å utvikle sine regneferdigheter gjennom praktisk anvendelse av reelle datasett som finnes i øvingsoppgavene. Her skal de bearbeide data for å oppdage sammenhenger, sammenligne resultater og presentere informasjon på ulike måter. I tillegg vil regneferdighetene bli styrket gjennom programmering, der elevene må velge riktige datatyper, utføre operasjoner, vurdere inndata og analysere resultater. De digitale ferdighetene vil også bli styrket gjennom bruk av et bredt spekter av programmer og verktøy, inkludert VS Code, PlantUML, Jupyter, Draw.io, Pencil, pdoc, pytest, git og flere Python-biblioteker.

### Tverrfaglig tema

Det tverrfaglige temaet i Informasjonsteknologi 2 er demokrati og medborgerskap. Vi tar opp dette temaet spesielt i bokene [1.10 Informasjonsteknologi](#), [2.10 Deepfakes](#), [3.10 Kunstig intelligens](#) og [4.10 Posisjonsdata](#), som dekker de to første kompetansemålene i læreplanen. I disse bokene utforsker vi hvordan bruk av informasjonsteknologi påvirker og utvikler samfunnet. Teknologi kan brukes og misbrukes, og vi står overfor etiske dilemmaer når fordelene med ny teknologi samtidig fører med seg negative konsekvenser. For eksempel kan posisjonsdata medføre overvåkning, kunstig intelligens kan føre til diskriminering, og Deepfakes kan spre falske nyheter. Dette innebærer mulige trusler for demokratiet. I tillegg legger vi vekt på [universell utforming](#) i bolken [4.1 Brukervennlighet](#), slik at alle har like muligheter til å delta aktivt i samfunnet og styrke sitt medborgerskap.

### Kjerneelementer

Dette læreverket er tilpasset kjerneelementene i faget, samtidig som bokene ofte overlapper flere av dem. *Kreativ problemløsning* blir ivaretatt gjennom flere runder, spesielt innenfor programmering og modellering. Elevene blir oppfordret til å bruke ulike metoder og teknikker for å utforske og løse problemstillinger, og vurdering av løsninger og forslag til forbedringer vektlagt. *Teknologi, individ og samfunn* blir også grundig behandlet i alle rundene. Bokene som [1.10 Informasjonsteknologi](#), [2.10 Deepfakes](#), [3.10 Kunstig intelligens](#) og [4.10 Posisjonsdata](#) omhandler hvordan informasjonsteknologi påvirker og endrer samfunnet. I tillegg diskuteres etiske spørsmål, personvern, universell utforming og informasjonssikkerhet. Kjernelementet *modellering og programmering* blir også utforsket gjennom flere runder. Elevene får lære om modellering av informasjonssystemer, algoritmisk tenkning og praktisk implementering av løsninger ved hjelp av programmering og utviklingsverktøy. Dette læreverket har også med bolker som omhandler *Digital representasjon og informasjonsutveksling*, som [1.6 Filer](#), [1.9 Reelle datasett med CSV](#), [2.1 Grafisk brukergrensesnitt med tkinter](#), [2.8 Utveksling av data med JSON](#), [2.9 Reelle datasett med JSON](#), [3.1 Jupyter](#), [NumPy og pandas](#), [3.2 Diagrammer med seaborn](#), [4.9 Reelle datasett med REST API](#), [4.5 Standarder for datahåndtering](#) og

## Smidig IT-2

### for eksklusiv bruk ved <navn på skole>

[5.11 API for web og mobil med Flask og Dash](#). Disse boklene gir elevene muligheten til å utforske digital representasjon og informasjonsutveksling på en praktisk og anvendbar måte.

#### **Overordnet del**

Et av formålene med den videregående opplæringen er å åpne dører mot verden og fremtiden, noe dette faget virkelig gjør<sup>37</sup>. Verdigrunnlaget blir også ivaretatt gjennom drøfting av etiske dilemmaer. Kompetansemål, grunnleggende ferdigheter og tverrfaglige tema har blitt diskutert tidligere. Samarbeid mellom skolen, hjemmet og lokalsamfunnet går utover hva vi dekker. Det samme gjelder lærerens rolle, selv om dette læreverket kan bidra til faglig oppdatering og tilpassing av undervisningen til elevenes behov. "Dybdelæring er å lære noe så godt at du forstår sammenhenger og kan bruke det du har lært i nye situasjoner", se [Dybdelæring](#). Vi har flere bolker som legger til rette for dybdelæring, for eksempel enhetstesting som bidrar til en dypere forståelse av programmering, modellering som utforsker ulike konsepter og deres innbyrdes sammenheng, samt bruk av reelle datasett og informasjonsteknologi i praksis. Disse aktivitetene gir elevene mulighet til å reflektere, utforske og anvende sin faglige kunnskap på en måte som fremmer dybdelæring.

Vurderingsseksempler

---

<sup>37</sup> [Overordnet del – verdier og prinsipper for grunnopplæringen](#)

# Smidig IT-2

## for eksklusiv bruk ved <navn på skole>

### 6.4 Kompetansemål

1. utforske og vurdere muligheter, utfordringer og konsekvenser ved bruk av informasjonsteknologi i ulike sammenhenger
2. drøfte etiske dilemmaer som oppstår som en konsekvens av informasjonsteknologi, både for individ og samfunn
3. utforske og vurdere alternative løsninger for design og implementering av et program
4. anvende objektorientert modellering til å beskrive et programs struktur
5. utvikle objektorienterte programmer med klasser, objekter, metoder og arv
6. vurdere og bruke strategier for feilsøking og testing av programkode
7. generalisere løsninger ved å utvikle og bruke gjenbrukbar programkode
8. vurdere brukervennligheten i egne og andres programmer og foreslå forbedringer
9. velge og bruke relevante systemutviklingsmetoder og -verktøy for samarbeid med andre
10. gjøre rede for standarder for lagring, utveksling og sikring av ulike typer data
11. bruke programmering til å innhente, analysere og presentere informasjon fra reelle datasett

Hvis vi ser dette dokumentet i Adobe Acrobat, kan vi taste **Alt + pil til venstre** for å komme tilbake til der hvor vi kom fra. Alternativt kan vi få fram **Forrige visning** på menylinja.

