Repetisjon/oppgaver i BED-1304 Python-Lab

Markus J. Aase

Forelesning 6 - Logikk og løkker

Informasjon

Til nå har vi gått igjennom flere ting i Python-lab. Det inkluderer temaer som:

- Python basics
- Funksjoner (Innebygde funksjoner og de vi definerer selv)
- Lister
- Dictionary
- Tuples
- Pakker som `Pandas` og `NumPy`

Videre skal vi fokusere (et av de kanskje viktigste) temaene i kurset - **Logikk og løkker**. I dette dokumentet vil dere kunne lese om:

- `if/else`-setninger
- `for`-løkker
- `while`-løkker

Dette dokumentet er lagd for å repetere og teste forståelsen av kjernepensum i **BED-1304 Python-Lab**. Husk at dette er et supplement til forelesning og seminar.

God koding!

Repetisjon og teori

Før vi går inn på tema i forelesning 6 *logikk og løkker*, må vi repetere kraften av å skrive en **god pseudokode**.

Pseudokode

Vi har tidligere i kurset diskutert hvordan vi går frem nå vi skal løse programmeringsoppgaver, og at *pseudokode* kan være en god hjelper.

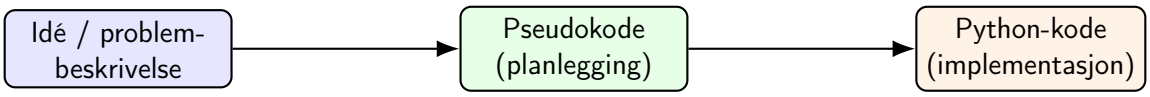
Definisjon

Pseudokode: er en forenklet og uformell måte å beskrive et dataprogram eller en algoritme på. Den skrives ofte i et språk som ligner på ekte programkode, men uten å følge de strenge syntaksreglene som et programmeringsspråk krever. Pseudokode brukes gjerne i planleggings- og designfasen sammen med flytskjemaer og andre metoder, og er ment som et hjelpemiddel for mennesker – ikke noe som kan kjøres direkte på en datamaskin.

Dette gjør det lettere å:

- fokusere på selve **problemløsningen**, uten å bli distrahert av detaljer i koden,
- **kommunisere algoritmer** til andre, også de som ikke kan Python,
- **planlegge koden trinnvis**, slik at overgangen til ekte Python blir mer effektiv og mindre feilutsatt,
- sammenligne og diskutere ulike løsninger før man bestemmer seg for implementering.

Kort sagt fungerer pseudokode som et **bindeledd mellom idé og ferdig Python-kode**, og er derfor et nyttig verktøy både i prosessen av å lære seg programmering, og drive med utvikling.



Figur 1: Pseudokode som bindeledd mellom idé og ferdig Python-kode.

Pseudokode - eksempel fra økonomi

Oppgavebeskrivelse: Vi ønsker å definere en funksjon som beregner hvordan en startkapital vokser med årlig rente over et gitt antall år. Formelen for sluttverdi er:

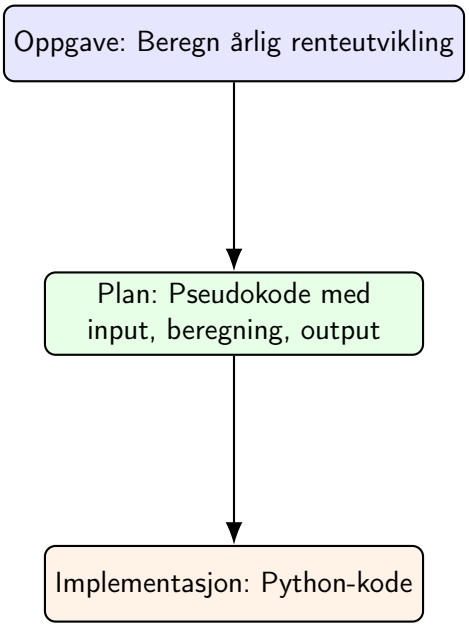
$$FV = K \cdot (1 + r)^n$$

der

- K = startkapital
- r = rente
- n = antall år
- FV = sluttkapital

Funksjonen skal ta inn disse parametrene og skrive ut resultatet i en f-string.

Skisse (prosess): Vi skisserer opp, *hva skal vi gjøre?*



Pseudokode: Vi fortsetter med å skrive en pseudokode, hvor vi stiller oss spørsmålet *hva er det vi skal kode*?

Pseudokode

- Definer en funksjon `future_value` med input: `capital, rate, years`.
- Beregn sluttverdi: `FV = capital * (1 + rate)**years`.
- Print ut en f-string: "Etter {years} år er verdien {FV}".

Nå har vi skissert opp hva vi skal gjøre, og hvordan vi har tenkt å kode dette. Da gjenstår det å skrive Python koden.

Python-kode:

```python
# Definerer funksjonen
def future_value(capital, rate, years):
    # Regner ut FV slik oppgaven beskriver
    FV = capital * (1 + rate)**years

    # Printer outputtet
    print(f"Etter {years} år er verdien {FV:.2f} kroner")

# Eksempelbruk av funksjonen med K = 10000 kr, r = 4% og years = 5 år
future_value(10000, 0.04, 5)
```

Logikk og løkker

I programmering er **logikk** og **løkker** grunnleggende byggesteiner. De lar oss ta beslutninger basert på betingelser, og gjenta handlinger mange ganger uten å måtte skrive den samme koden flere ganger. Her ser vi kjapt på `for`-løkker, `if/else`-setninger, og `while`-løkker.

If/else setning

Forklaring: Med `if/else` kan vi sjekke en betingelse, og utføre kode kun hvis betingelsen er sann. Dersom betingelsen ikke er sann, kan vi bruke `else` til å kjøre en alternativ handling.

Eksempel i Python:

```python
x = 10
if x > 5:
    print("x er større enn 5")
else:
    print("x er 5 eller mindre")
```

For-løkker

Forklaring: En `for`-løkke lar oss gjenta en blokk kode et bestemt antall ganger eller iterere over elementene i en samling (f.eks. en liste).

Eksempel i Python:

```python
for i in range(1, 5):
    print(f"Tallet er {i}.")
```

While-løkker

Forklaring: En `while`-løkke kjører så lenge en betingelse er sann. Dette er nyttig når vi ikke vet hvor mange ganger koden skal gjentas.

Eksempel i Python:

```python
x = 0
while x < 5:
    print(f"x er nå {x}")
    x += 1 # Samme som å skrive x = x + 1
```

Oppsummering:

- `if/else` brukes til å ta valg.
- `for`-løkker brukes til å gjenta en handling et fast antall ganger eller gjennom en liste.
- `while`-løkker brukes når vi ønsker gjentakelse inntil en betingelse ikke lenger er oppfylt.

If-else setning - grundigere forklart

Definisjon: If/else-betingelse

En **if/else**-setning brukes i programmering for å utføre ulike kodeblokker avhengig av om en logisk betingelse er sann (`True`) eller falsk (`False`).
Generell syntaks i Python:

```
if betingelse:
    # kode som kjører hvis betingelsen er sann (True)
else:
    # kode som kjører hvis betingelsen er falsk (False)
```

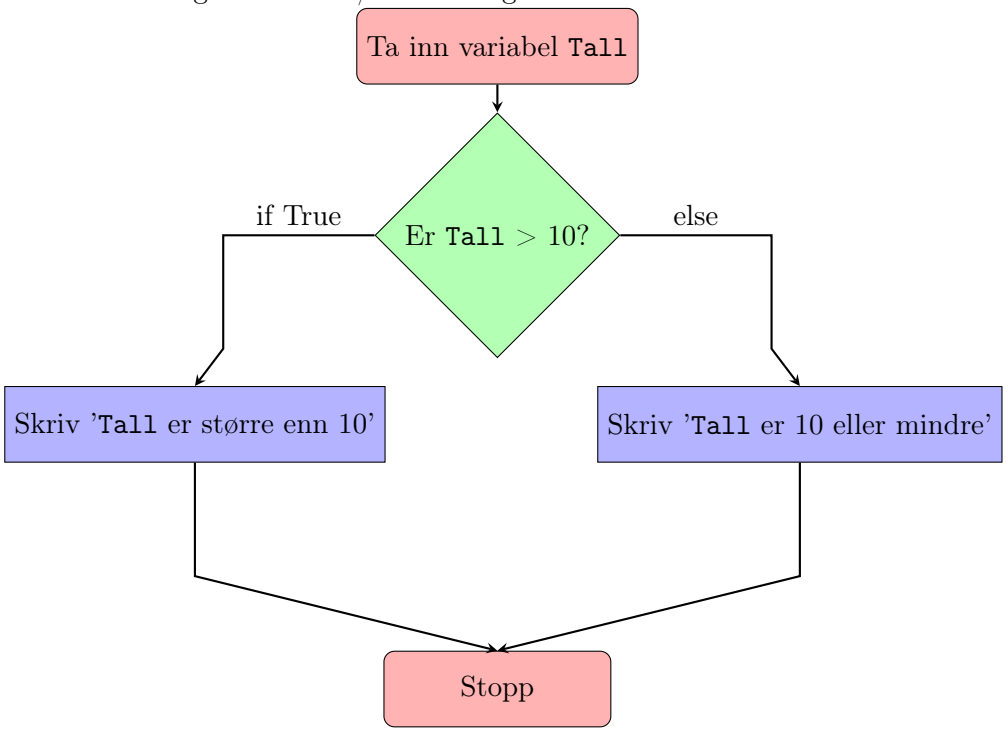
Eksempel med flytdiagram

La oss si at vi får følgende i oppgave:

Oppgave

Ta inn en variabel `Tall`, og sjekk om tallet er over eller under 10.

Under gir vi et enkelt eksempel, hvor vi sjekker om en variabel `Tall` er større enn 10. Dette illustrerer logikken i en if/else-setning:



Dette eksempelet kan kanskje virke unødvendig, eller at dere tenker "*hvorfor skal vi lære dette*?"

Men faktisk kan det være ganske nyttig med enkle **if/else-setninger**.

Økonomisk eksempel

I Norge (i 2025), er det en **frikortgrense** på 100 000 kr. Det betyr at, tjener du mindre enn 100 000 kr, trenger du ikke å betale skatt. Tjener du mer enn dette, **må** du betale skatt.

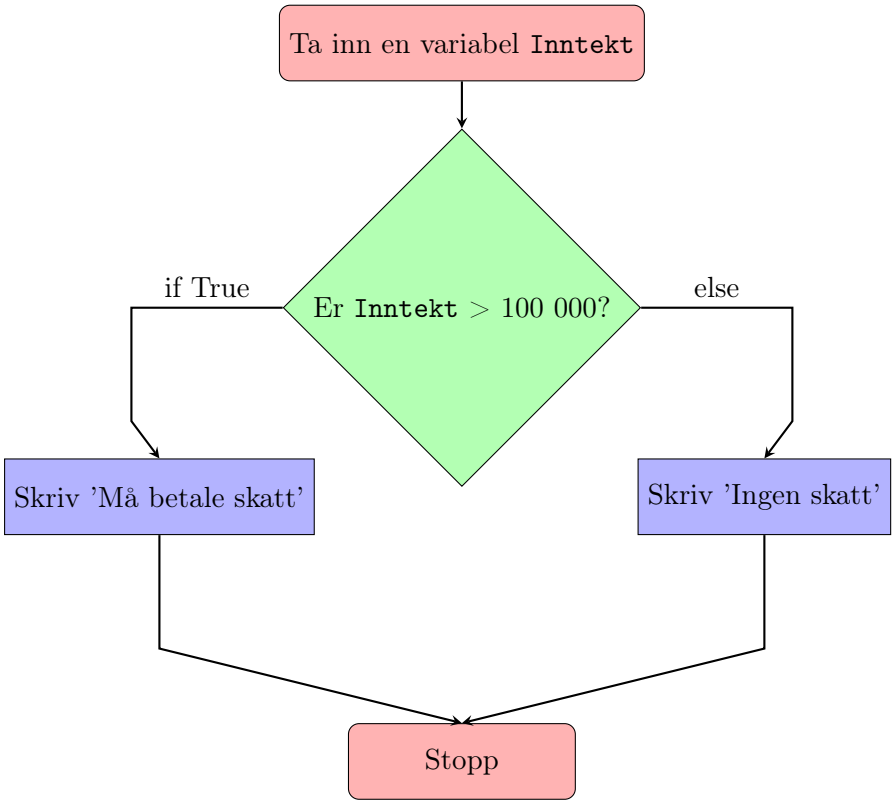
Dette er jo en **if/else-setning** i sin natur! La oss se på hvordan vi ville løst det ved hjelp av pseudokode, skisser og til slutt, hvordan vi kan skrive det som Python-kode.

Oppgave: Ta inn inntekt, og sjekk om personen må betale skatt eller ikke.

Pseudokode

- Les inn inntekt `Inntekt`
- Hvis `Inntekt` $>$ 100000, skriv 'Må betale skatt'
- Ellers, skriv 'Ingen skatt'

Flytdiagram



Python-kode

Her er hvordan vi kan implementere eksemplet i Python:

```
# Leser inn inntekt fra bruker
inntekt = float(input("Skriv inn din inntekt: "))

# If/else-betingelse
if inntekt > 100000:
    print("Må betale skatt")
else:
    print("Ingen skatt")
```

Forklaring:

- 'input()' leser inn verdien fra brukeren.
- 'float()' gjør teksten om til tall slik at vi kan sammenligne den.
- 'if' sjekker om inntekten er over 100 000.
- 'else' håndterer tilfellene der inntekten er lik eller mindre enn 100 000.

Vi har flere måter å sjekke betingelser på

I `if`-setninger kan vi bruke ulike typer betingelser for å bestemme hvilke kodeblokker som skal kjøres.

Sammenligninger

Sammenligningsoperatorer brukes til å sammenligne verdier:

- `==` : lik
- `!=` : ikke lik
- `>` : større enn
- `<` : mindre enn
- `>=` : større enn eller lik
- `<=` : mindre enn eller lik

Eksempel:

```
alder = 18

# Hvis alder er større eller lik 18
if alder >= 18:
    print("Stemmerett")
else:
    print("Ikke stemmerett")
```

Logiske operatorer

Logiske operatorer brukes til å kombinere flere betingelser:

- `and` : sann hvis begge betingelser er sanne
- `or` : sann hvis minst én betingelse er sann
- `not` : inverterer sannhetsverdien

Eksempel:

```
alder = 20
har_lappen = True

# Hvis alder er over eller lik 18, og har lappen
if alder >= 18 and har_lappen:
    print("Kan kjøre bil")
else:
    print("Kan ikke kjøre bil")
```

Når if/else ikke er nok - da trenger vi elif!

Noen ganger holder det ikke med bare **if** og **else**. For eksempel, hvis vi lager et program hvor brukeren skriver inn sin alder, og vi ønsker å plassere personen i en av flere kategorier:

- **Barn** (0–12 år)
- **Ungdom** (13–17 år)
- **Voksen** (18–64 år)
- **Honnør** (65+ år)

Her må vi sjekke flere betingelser i rekkefølge. Dette kan vi gjøre med `elif` (else if). `elif` lar oss lage flere grener før vi eventuelt bruker en `else`-blokk til å håndtere alt som ikke matcher de tidligere betingelsene. Dette kan vi kalle en **if/elif/else** setning (eller betingelse).

Definisjon: If/elif/else-betingelse

Definisjon av if/elif/else setning: En **if/elif/else**-setning brukes i programmering for å utføre ulike kodeblokker, når vi har en logisk betingelse med **mer enn to utfall**.
Generell syntaks i Python:

```
if betingelse1:
    # kode som kjører hvis betingelse1 er sann
elif betingelse2:
    # kode asom kjører hvis betingelse2 er sann
else:
    # kode som kjører hvis ingen av de overstående betingelsene er sanne
```

Hvis vi prøver å løse eksempelet over, med ulike alderskategorier - kan det gjøres på følgende vis.

```
# Leser inn alder fra bruker
alder = int(input("Skriv inn din alder: "))

# Sjekker hvilken kategori personen tilhører
if alder <= 12:
    print("Barn")
elif alder <= 17:
    print("Ungdom")
elif alder <= 64:
    print("Voksen")
else:
    print("Honnør")
```

Forklaring:

- ‘input()‘ leser inn verdien fra brukeren.
- ‘int()‘ konverterer tekst til heltall slik at vi kan sammenligne alder.
- ‘if alder <= 12:‘ sjekker om personen er 12 år eller yngre. Hvis ja, skrives ’Barn’.
- ‘elif alder <= 17:‘ sjekker neste betingelse (13–17 år). Hvis sann, skrives ’Ungdom’.
- ‘elif alder <= 64:‘ sjekker om alder er mellom 18 og 64 år. Hvis sann, skrives ’Voksen’.
- ‘else:‘ fanger opp alle aldre 65 og eldre, og skriver ’Honnør’.

Økonomisk eksempel med lister og if/else-setninger

I økonomi og personlig finans kan vi bruke lister for å kategorisere ulike typer inntekter, skatter eller avgifter. For eksempel kan vi ha ulike typer skattepliktige og ikke-skattepliktige inntekter, og bruke en if/else-struktur for å sjekke kategorien til en inntekt.

```
# Eksempel på inntektskategorier
skattepliktig = ['lønn', 'bonus', 'kapitalinntekt']
skattefri = ['gave', 'stipend', 'frikort']

# En inntektstype vi vil sjekke
inntekt = 'bonus'

# Sjekker hvilken kategori inntekten tilhører
if inntekt in skattepliktig:
    print(f"{inntekt} er skattepliktig")
elif inntekt in skattefri:
    print(f"{inntekt} er skattefri")
else:
    print(f"Kategorien til {inntekt} er ukjent")
```

Forklaring:

- Vi definerer to lister: `skattepliktig` og `skattefri`.
- Variabelen `inntekt` sjekkes mot listene ved hjelp av `in`.
- `if inntekt in skattepliktig:` kjører hvis inntekten er i den skattepliktige listen.
- `elif inntekt in skattefri:` kjører hvis inntekten finnes i den skattefrie listen.
- `else` fanger opp alle inntekter som ikke finnes i noen av listene.

For-løkker - grundigere forklart

Definisjon: For-løkke

En **for-løkke** brukes i programmering for å gjenta en kodeblokk et bestemt antall ganger, eller over elementene i en liste, et intervall eller en sekvens.
Generell syntaks i Python:

```
for element in sekvens:
    # kode som kjører for hvert element
```

Eksempel med oppgave

La oss si vi ønsker å skrive ut tallene fra 1 til 5.

Oppgave

Bruk en for-løkke til å skrive ut tallene 1, 2, 3, 4 og 5.

Python-kode

```
# Skriver ut tallene fra 1 til 5
for i in range(1, 6):
    print(i)
```

Forklaring:

- 'range(1, 6)' genererer tallene 1, 2, 3, 4, 5.
- For hver verdi av 'i' i sekvensen kjører koden i løkken.
- 'print(i)' skriver ut verdien av 'i' hver gang løkken kjører.
- Legg merke til at tallet 6 ikke skrives ut, fordi funksjonen `range(a, b)` teller **fra og med** `a`, og **opp til** (men ikke inkludert) `b`.

Tips: Prøv selv, og gjør det kjent med løkker.

For-løkke over en liste

Vi kan også iterere over elementene i en liste:

```
frukt = ["eple", "banan", "appelsin"]

for f in frukt:
    print(f)
```

Forklaring:

- Løkken kjører én gang for hvert element i listen 'frukt'.
- Variabelen 'f' tar verdien til hvert element underveis i løkken.
- Her vil den printe følgende:
  - *eple*
  - *banan*
  - *appelsin*

While-løkker - grundigere forklart

Definisjon: While-løkke

En **while-løkke** brukes i programmering for å gjenta en kodeblokk så lenge en gitt betingelse er sann (`True`). Løkken fortsetter å kjøre inntil betingelsen blir falsk (`False`).
Generell syntaks i Python:

```
while betingelse:
    # kode som kjører så lenge betingelsen er sann
```

Eksempel med oppgave

La oss si vi ønsker å skrive ut tallene fra 1 til 5 ved hjelp av en while-løkke.

Oppgave

Bruk en while-løkke til å skrive ut tallene 1, 2, 3, 4 og 5.

Python-kode

```python
# Skriver ut tallene fra 1 til 5
i = 1
while i <= 5:
    print(i)
    i += 1  # øk i med 1 hver gang løkken kjører
```

Forklaring:

- Vi starter med å sette `i = 1`.
- Løkken kjører så lenge `i <= 5` er sann.
- `print(i)` skriver ut verdien av `i`.
- `i += 1` øker `i` med 1 for hver iterasjon. Hvis vi ikke gjør dette, vil `while`-løkka kjøre for alltid...
- Når `i` blir 6, blir betingelsen `i <= 5` falsk, og løkken stopper.

While-løkke med liste

Vi kan også bruke while-løkker til å iterere over en liste:

```python
frukt = ["eple", "banan", "appelsin"]
i = 0

while i < len(frukt):
    print(frukt[i])
    i += 1
```

Forklaring:

- Vi bruker en indeks `i` for å holde styr på posisjonen i listen.
- `len(frukt)` gir lengden på listen.
- Løkken kjører så lenge `i` er mindre enn lengden på listen.
- `frukt[i]` henter elementet på posisjon `i`, og `i += 1` flytter til neste element.
- Dette vil printe:
  - *eple*
  - *banan*
  - *appelsin*

Økonomisk eksempel: Akkumulere inntekt med `while`-løkke

Oppgave: Vi ønsker å finne ut hvor mange måneder det tar å oppnå en total inntekt på minst 30 000 kr, gitt månedlige inntekter:
`[3000, 3200, 3500, 2800, 3000, 3100, 3300, 3400, 3600, 3700, 3500, 3400]`
Dette kan vi løse i Python ved hjelp av en `while`-løkke.

```
monthly_income = [3000, 3200, 3500, 2800, 3000, 3100, 3300, 3400, 3600, 3700,
    3500, 3400]
total_income = 0
index = 0

while total_income < 30000 and index < len(monthly_income):
    total_income += monthly_income[index]  # Legger til inntekten for den
    gjeldende måneden
    index += 1                             # Går til neste måned

print(f"Total inntekt nådd: {total_income} kr etter {index} måneder.")
```

Forklaring av koden

- **monthly_income**: Liste med månedlig inntekt for 12 måneder.
- **total_income**: Starter på 0, akkumulerer inntektene.
- **index**: Holder styr på hvilken måned vi er på i listen.
  - Legg merke til at vi må starte med en verdi av `index` (tredje kodelinje: `index = 0`), og den defineres **før** `while`-løkka starter.

While-løkke:

- `while total_income < 30000 and index < len(monthly_income):` Løkken kjører så lenge total_income er mindre enn 30 000 kr og vi ikke har gått gjennom alle månedene.
- Inne i løkken:
  - `total_income += monthly_income[index]`: Legger til inntekten for den gjeldende måneden.
  - `index += 1`: Øker indeksen for å gå videre til neste måned.

Utskrift: Etter løkken er ferdig, altså når betingelsen går fra `True` til `False`, skriver vi ut den totale inntekten som er oppnådd og etter hvor mange måneder.

Komponenter i `while`-løkker

Komponent	Beskrivelse	Eksempel
`while`	Starter løkken og definerer betingelsen for å fortsette.	`while total_income < 30000:`
Betingelse	Uttrykk som evalueres til `True` eller `False`.	`total_income < 30000`
Løkkeblokk	Koden som kjøres så lenge betingelsen er sann.	`total_income += monthly_income[index]`
`break`	Avslutter løkken når det ikke er nødvendig å fortsette.	`if total_income >= 30000: break`
`continue`	Hopper over resten av løkkens kode for den nåværende iterasjonen.	`continue`
`index`	Indeks, for å gi `while`-løkka et startpunkt og sikre at løkka ikke kjører for alltid.	`index = 0` (før løkka) `index +=1` (inni løkka)

Del 1: Flervalgsoppgaver - Logikk og løkker

1. Hva vil følgende kode printe?

```
x = 7
if x > 5:
    print("Større enn 5")
else:
    print("5 eller mindre")
```

   a) Større enn 5
   b) 5 eller mindre
   c) Ingenting
   d) Feilmelding

2. Hvilket av følgende utsagn er sant om en for-løkke?

   a) Den kan kun iterere over tall
   b) Den kan iterere over lister, strenger og range
   c) Den kjører alltid kun én gang
   d) Den fungerer bare med if/else

3. Hva gjør denne koden?

```
frukt = ["eple", "banan", "appelsin"]
for f in frukt:
    print(f)
```

   a) Printer navnet på listen
   b) Printer hvert element i listen
   c) Teller antall elementer
   d) Ingenting

4. Hvilket av disse er riktig måte å lage en while-løkke som teller fra 1 til 3?

   a)
```
i = 1
while i <= 3:
    print(i)
    i += 1
```

   b)
```
while i in range(1, 4):
    print(i)
```

   c)
```
for i in range(1, 4):
    print(i)
```

   d)
```
while i < 3:
    print(i)
```

5. Hvilken operator brukes for å kombinere to betingelser i en if-setning?

   a) +
   b) and
   c) %
   d) ==

6. Hva vil følgende kode printe?

```
alder = 16
if alder >= 18:
    print("Voksen")
elif alder >= 13:
    print("Ungdom")
else:
    print("Barn")
```

   a) Voksen
   b) Ungdom
   c) Barn
   d) Feilmelding

7. Hva gjør `if x in liste:` i Python?

   a) Sjekker om variabelen x finnes i listen
   b) Legger til x i listen
   c) Fjerner x fra listen
   d) Gir alltid True

Del 2: Hva blir output? - Logikk og løkker

1.
```
x = 3
if x > 0 and x < 5:
    print("Innenfor intervallet")
else:
    print("Utenfor intervallet")
```

2.
```
frukt = ["eple", "banan", "appelsin"]
for f in frukt:
    print(f.upper())
```

3.
```
i = 0
while i < 3:
    print(i)
    i += 1
```

4.
```
alder = 20
har_lappen = False

if alder >= 18 and har_lappen:
    print("Kan kjøre bil")
else:
    print("Kan ikke kjøre bil")
```

5.
```
inntekt = 120000
if inntekt > 100000:
    print("Må betale skatt")
else:
    print("Ingen skatt")
```

Del 3: Praktiske oppgaver - Logikk og løkker

1. Lag en funksjon `klassifiser_alder` som tar inn alder og returnerer 'Barn', 'Ungdom', 'Voksen' eller 'Honnør' basert på aldersgrupper:
   - Barn: 0–12
   - Ungdom: 13–17
   - Voksen: 18–64
   - Honnør: 65+
2. Lag en liste med tall `[2, 5, 8, 11, 14]` og bruk en for-løkke til å printe kun de tallene som er større enn 5.
3. Lag en while-løkke som teller ned fra 10 til 1 og skriver ut hvert tall.
4. Lag en liste `inntekter = [50000, 120000, 95000, 200000]` og bruk en for-løkke med if/else til å skrive ut om hver inntekt må betale skatt (frikortgrense 100000).
5. Lag en funksjon `summer_liste(liste)` som bruker en while-løkke til å summere alle elementene i en liste og returnerer summen.

Del 4: En utfordring - Logikk og løkker

Anta at du har en startkapital på 50 000 kr og planlegger å investere i aksjer over en periode på 12 måneder. Hver måned kan aksjemarkedet enten gå opp eller ned, og du får en tilfeldig månedlig avkastning mellom -5% og +10%. I tillegg har du faste månedlige utgifter på 3 000 kr.

Oppgave: Skriv et Python-program som simulerer kapitalutviklingen din over 12 måneder. Programmet skal:

1. Bruke `if/else` til å sjekke om kapitalen etter utgifter blir negativ. Hvis den blir negativ, stopp programmet og skriv ut en **advarsel om konkurs**.
2. Bruke en `while`-løkke til å summere kapitalen måned for måned inntil en målkapital på 70 000 kr er nådd, eller til pengene tar slutt.
3. Skrive ut kapitalen ved slutten av hver måned og totalt antall måneder simulert.

Tips: Bruk `random.uniform(-0.05, 0.10)` for å generere månedlig avkastning. Da må du først skrive:

```
import random
```

Kjører du funksjonen:

```
variabel_a =random.uniform(-0.05, 0.10
variabel_a
```

vil `variabel_a` gi et tilfeldig tall mellom -0.05 og 0.10.

Lykke til!

Utfordring - del 2

Hvis du har gjort oppgaven over, hva skjer om du ikke tar høyde for månedlige utgifter?

Del 5: Vanskelig!

OBS: Denne oppgaven er ment for deg som ønsker en utfordring. For å løse denne oppgaven anbefales det å starte med pseudokode, tegne en skisse over programflyten og kommentere koden nøye. Da får du best læringsutbytte.

Oppgavetekst

Du skal lage et lite program som hjelper en student med å holde oversikt over økonomien sin. Studenten har en månedlig inntekt, faste utgifter (husleie, strøm, mat osv.), og ønsker å sette av penger til sparing. I tillegg kan det dukke opp tilfeldige ekstrautgifter (for eksempel reparasjon av sykkel eller en uventet regning).

- Bruk en **dictionary** til å lagre faste utgifter (f.eks. `utgifter = {"husleie": 6000, "mat": 3000, 'strøm': 800}`).
- Bruk en **liste** til å simulere månedlige ekstrautgifter (for eksempel `[500, 1200, 0, 800]`, hvor 500 er for måned 1, 1200 er for måned 2 og så videre).
- Bruk en **for-løkke** eller en **while-løkke** til å simulere flere måneder fremover (altså, lengden av månedlige ekstrautgifter-lista).
- Bruk **if/else** til å sjekke om studenten klarer å spare penger hver måned, eller om kontoen går i minus.
- Lag en **funksjon** som tar inn `månedlig inntekt`, `utgifter` og `ekstrautgifter`, og returnerer hvor mye som er spart (eller tapt) etter alle månedene.
- (Ekstra vanskelig) Bruk **NumPy** til å regne ut gjennomsnittlig sparing per måned.
- (Ekstra vanskelig++) Bruk **pandas** til å lagre dataene i en tabell med kolonner for `måned`, `inntekt`, `utgifter`, `sparing`, og print tabellen pent til slutt.

Tips til løsningsstrategi

1. Start med å skrive pseudokode (f.eks. hva skjer først, hva skal løkken gjøre, hvordan sjekker du saldoen).
2. Lag en enkel versjon først (uten pandas/numpy).
3. Utvid deretter programmet gradvis.

Mål med oppgaven

Denne oppgaven trener deg i å kombinere flere Python-verktøy (funksjoner, løkker, betingelser, lister, dictionaries, numpy/pandas) i en økonomisk kontekst. Målet er å lære å bygge et lite, men godt program for studentøkonomi.

Løsningsforslag - Logikk og løkker

Del 1: Flervalgsoppgaver

1: a, 2: b, 3: b, 4: a, 5: b, 6: b, 7: a

Del 2: Hva blir output?

1:

```
Innenfor intervallet
```

2:

```
EPLE
BANAN
APPELSIN
```

3:

```
0
1
2
```

4:

```
Kan ikke kjøre bil
```

5:

```
Må betale skatt
```

Del 3: Praktiske oppgaver - Løsningsforslag

Oppgave 1

```python
def klassifiser_alder(alder):
    if alder <= 12:
        return "Barn"
    elif alder <= 17:
        return "Ungdom"
    elif alder <= 64:
        return "Voksen"
    else:
        return "Honnør"

# Eksempel
print(klassifiser_alder(20))  # Voksen
```

Oppgave 2

```python
# Vår liste med tall
tall = [2, 5, 8, 11, 14]

# for-løkke gjennom lista
for t in tall:
    # Hvis t > 5, så printer vi t
    if t > 5:
        print(t)
    # Hvis t < 5, så gjør vi ingenting
```

Oppgave 3

```python
# Starter med i = 10
i = 10
while i > 0:
    print(i) # printer i
    i -= 1   # teller ned 1 steg fra forrige i-verdi
```

Oppgave 4

```python
inntekter = [50000, 120000, 95000, 200000]
for inntekt in inntekter:
    if inntekt > 100000:
        print(f"{inntekt} kr: Må betale skatt")
    else:
        print(f"{inntekt} kr: Ingen skatt")
```

Oppgave 5

```python
# Definerer funksjonen summer_liste med parameter 'liste'
def summer_liste(liste):
    i = 0
    total = 0
    while i < len(liste):
        total += liste[i]
        i += 1
    return total

# Eksempel, hvor vi bruker [1,2,3,4] som argument (altså lista vår).
print(summer_liste([1,2,3,4]))  # 10 (fordi 1 + 2 + 3 + 4 = 10)
```

Løsningsforslag - Utfordring

Her er det mange veier til Rom. Dette er én måte å løse oppgaven på.

```python
import random

# Startkapital og parametere
capital = 50000
monthly_expense = 3000
goal = 70000
month = 0

print("Starter simuleringen...")

# While-løkke for å nå målkapital
while capital > 0 and capital < goal and month <= 12:
    month += 1
    monthly_return = random.uniform(-0.05, 0.10)  # tilfeldig avkastning
    capital = capital * (1 + monthly_return) - monthly_expense

    if capital <= 0:
        print(f"Konkurs etter {month} måneder!")
        break

    print(f"Måned {month}: Kapital = {capital:.2f} kr")

if capital >= goal:
    print(f"Målkapital på {goal} kr nådd etter {month} måneder!")
```

Forklaring:

- `while capital > 0 and capital < goal and month <= 12:` sikrer at simuleringen stopper ved konkurs, eller når målkapital er nådd, eller det har gått 12 måneder.
- `if capital <= 0:` sjekker om du går konkurs og avslutter simuleringen.
- `month += 1` øker verdien av `month` med 1 per gang vi gjør beregningen. Når den blir 12 stoppes løkka.
- Til slutt skriver programmet ut kapitalen hver måned og totalt antall måneder.

Løsningsforslag - utfordring del 2

Fjerner du `monthly_expense` i løsningen over, inne i while-løkka, vil du se at du oftere når kapital-målet.

Noe som gir fullstendig mening, siden du har kvitta deg med 3 000 kr i utgifter.

Løsningsforslag - del 5

```python
# Funksjon for å simulere økonomien
def simulate_budget(income, fixed_expenses, extra_expenses):
    savings = 0
    month = 0

    # Summerer de faste utgiftene (dictionary values)
    total_fixed = sum(fixed_expenses.values())

    # Gå gjennom alle måneder (basert på listen med ekstrautgifter)
    # Lengden på extra_expenses bestemmer hvor mange iterasjoner av for-løkka vil kjøres
    for i in extra_expenses:
        month += 1
        total_expenses = total_fixed + i
        balance = income - total_expenses
        savings += balance

        if balance >= 0:
            print(f"Måned {month}: Sparte {balance} kr")
        else:
            print(f"Måned {month}: Gikk {abs(balance)} kr i minus")

    print(f"\nTotalt spart etter {month} måneder: {savings} kr")
    return savings

# Eksempelbruk
income = 15000 # Studiestøtte + en ekstra jobb
fixed_expenses = {"husleie": 6000, "mat": 3000, "strøm": 800}
# Ekstra utgifter i månedene fremover
extra_expenses = [500, 1200, 0, 800, 400, 3000]

simulate_budget(income, fixed_expenses, extra_expenses)
```

Forklaring:

- `fixed_expenses` er en dictionary med faste kostnader.
- `extra_expenses` er en liste som simulerer uforutsette utgifter.

- For-løkken går måned for måned, regner ut saldo og legger det til sparingen.
- If/else sjekker om saldoen er positiv eller negativ.

```python
import numpy as np
import pandas as pd

def simulate_budget_pandas(income, fixed_expenses, extra_expenses):
    total_fixed = sum(fixed_expenses.values())
    months = len(extra_expenses)

    # Definerer en dictionary med nøkler, men tomme lister som verdier
    # Denne skal bli til en Pandas Dataframe senere
    data = {"Måned": [], "Inntekt": [], "Utgifter": [], "Sparing": []}
    total_savings = 0

    # For løkke gjennom månedene
    for i in range(months):
        expenses = total_fixed + extra_expenses[i]
        saving = income - expenses
        total_savings += saving

        # Legg til data i 'data' dictionary-en
        data["Måned"].append(i+1)
        data["Inntekt"].append(income)
        data["Utgifter"].append(expenses)
        data["Sparing"].append(saving)

    # Gjør om dictionary-en kalt 'data' til en dataframe 'df'.
    df = pd.DataFrame(data)

    print(df)
    print("\nGjennomsnittlig sparing per måned:", np.mean(df["Sparing"]))
    print("Total sparing:", total_savings)

    return df

# Eksempelbruk
income = 15000
fixed_expenses = {"husleie": 6000, "mat": 3000, "strøm": 800}
extra_expenses = [500, 1200, 0, 800, 400, 0, 700, 6000]

simulate_budget_pandas(income, fixed_expenses, extra_expenses)
```

Forklaring:

- Vi bruker en `pandas.DataFrame` til å lagre alle månedene i en tabell.
- `NumPy` brukes til å regne ut gjennomsnittlig sparing per måned.
- Tabellen gir en ryddig oversikt over økonomien måned for måned.