# 1 Betingelser (avansert)

I virkelige situasjoner kan betingelser vi må implementere i koden vår avhenge av flere faktorer. I denne seksjonen vil vi se verktøy for å kombinere disse betingelsene og verktøy for å sikre at vi kombinerer dem på riktig måte.

## 1.1 Logiske operatorer: `and`, `or`, `not`, osv.

Vi kan ha mer avanserte betingelser ved å bruke logiske operatorer.

- `and`: returnerer `True` hvis **alle** betingelsene som den kobler sammen, er sanne. Ellers, vil den returnere `False`.
- `or`: returnerer `True` hvis **minst én** av betingelsene som den kobler sammen, er sann. Ellers, vil den returnere `False`.
- `not`: evaluerer omvendt av den opprinnelige betingelsen. Det vil si at den returnerer `True` hvis betingelsen var `False` og `False` hvis den var `True`

[1]:
```
a = True
b = False

# Syntaksen f"{uttrykk = }" skiver både uttrykket og evalueringen (resultatet)
# av uttrykket
print(f"a:              {a}")
print(f"b:              {b}")
print(f"not a:          {not a}")
print(f"not b:          {not b}")
print(f"a and b:        {a and b}")
print(f"a or b:         {a or b}")
print(f"not (a or b):   {not (a or b)}")
```

```
a:              True
b:              False
not a:          False
not b:          True
a and b:        False
a or b:         True
not (a or b):   False
```

## 1.2 Bruken av funksjoner for å teste koden vår bedre

Det er en god praksis å strukturere koden vår med funksjoner. Dette forbedrer lesbarheten, fleksibiliteten og forenkler fremtidig vedlikehold av koden.

[2]:
```
karakter = 34

def eksamen(karakter):
    print(f"Du fikk {karakter}")
    # if 1
    # k >= 50: besto
    if karakter >= 50:
        print("Du besto")
    # k <  50: strøk
    else:
```

```
        print("Du strøk")
        # - if 2
        # - k >= 25: prøv igjen
        if karakter >= 25:
            print("Du kan prøve igjen")
        # - k <  25: kan ikke prøve igjen
        else:
            print("Du kan ikke prøve igjen")
```

I tillegg, nå som funksjonen er definert, er det enkelt å prøve alle tilfeller ved å kalle på funksjonen med ulike parametre. Å kalle på funksjonen med ulike parametre er en god måte å *teste* koden vår og å sørge for at vi ikke har gjort en logiske feil.

1. Tenk på flyten av en gitt inndata i koden din koden er utført. Prøv med forskjellige inndata i tankene dine og noter det forventet resultatet. Prøv å tenke på typiske inndata man ville bruke med denne koden, men også kanttilfeller som negative tall, 0 osv.
2. Test koden din ved å kjøre den med denne inndata. Sjekk at resultatet er som forventet.

[3]:
```
eksamen(62)
```

```
Du fikk 62
Du besto
```

[4]:
```
eksamen(32)
```

```
Du fikk 32
Du strøk
Du kan prøve igjen
```

[5]:
```
eksamen(18)
```

```
Du fikk 18
Du strøk
Du kan ikke prøve igjen
```

[6]:
```
def stor_rabatt(medlem, antall_artikler):
    print(f"Medlem: {medlem} Antall artikler kjøpt: {antall_artikler}")
    if medlem and (antall_artikler >= 3):
        print("Du får 60 prosent rabatt")
```

[7]:
```
stor_rabatt(True, 2)
stor_rabatt(True, 3)
stor_rabatt(False, 1)
stor_rabatt(False, 4)
```

```
Medlem: True Antall artikler kjøpt: 2
Medlem: True Antall artikler kjøpt: 3
Du får 60 prosent rabatt
Medlem: False Antall artikler kjøpt: 1
Medlem: False Antall artikler kjøpt: 4
```

[8]:
```
def rabatt(medlem, antall_artikler):
    print(f"Medlem: {medlem} Antall artikler kjøpt: {antall_artikler}")
    if medlem or (antall_artikler >= 3):
        print("Du får 30 prosent rabatt")
```

[9]:
```
rabatt(True, 2)
rabatt(True, 3)
rabatt(False, 1)
rabatt(False, 4)
```

```
Medlem: True Antall artikler kjøpt: 2
Du får 30 prosent rabatt
Medlem: True Antall artikler kjøpt: 3
Du får 30 prosent rabatt
Medlem: False Antall artikler kjøpt: 1
Medlem: False Antall artikler kjøpt: 4
Du får 30 prosent rabatt
```

I eksempelet nedenfor ønsker vi å flytte med en flyttebil. Kan flyttebilen ta alt på en gang? Det kan den hvis bilen er større enn pakkene som skal flyttes. Vi legger til tilfellet "Bør vi ta en mindre bil hvis bilen var altfor stor?"

[10]:
```
def flyttebil(pakker_volum, bil_max_volum):
    mulig = bil_max_volum >= pakker_volum
    if not mulig:
        print("Vi kan ikke ta alt på en gang.")
    else:
        print("Vi kan ta alt!")
        if bil_max_volum > 3*pakker_volum:
            print("Vi bør ta en MYE mindre bil.")
        elif bil_max_volum > 2*pakker_volum:
            print("Vi bør ta en litt mindre bil.")
        else:
            print("Vi tok den riktige størrelsen.")

    return mulig

mulig = flyttebil(15, 15)
```

```
Vi kan ta alt!
Vi tok den riktige størrelsen.
```

[11]:
```
mulig = flyttebil(10, 15)
```

```
Vi kan ta alt!
Vi tok den riktige størrelsen.
```

[12]:
```
mulig = flyttebil(7, 15)
```

```
Vi kan ta alt!
Vi bør ta en litt mindre bil.
```

[13]:
```
mulig = flyttebil(3, 15)
```

```
Vi kan ta alt!
Vi bør ta en MYE mindre bil.
```

### 1.3 Avansert while løkke

Dette eksempelet er litt mer avansert fordi vi trenger å initialisere 2 variabler og oppdateringen avhenger av disse 2 variabler. I tillegg, er det 3 handlinger i løkken.

[14]:
```
# Opprett variabelen før løkken
total = 0
i = 1
print(f"Før løkken. i: {i}, total: {total}")

# Fortsett å legge til tall sammen så lenge den totale summen er mindre enn 100.
# 0 + 1 + 2 + 3 + 4 + ... + i - 1 + i
while total < 100:

    # Oppdater total sum (knyttet til betingelsen)
    total = total + i
    print(f"Iterasjon: {i}, total: {total}")

    # Oppdater iterasjonsvariabelen
    i = i + 1

print(f"Etter løkken. i: {i}, total: {total}")
```

```
Før løkken. i: 1, total: 0
Iterasjon: 1, total: 1
Iterasjon: 2, total: 3
Iterasjon: 3, total: 6
Iterasjon: 4, total: 10
Iterasjon: 5, total: 15
Iterasjon: 6, total: 21
Iterasjon: 7, total: 28
Iterasjon: 8, total: 36
Iterasjon: 9, total: 45
Iterasjon: 10, total: 55
Iterasjon: 11, total: 66
Iterasjon: 12, total: 78
Iterasjon: 13, total: 91
Iterasjon: 14, total: 105
Etter løkken. i: 15, total: 105
```