1 Boolsk verdier, betingelser og logiske operatorer

Betingelser lar oss skrive programmer som kan tilpasse seg forskjellige situasjoner ved å utføre tilpassede handlinger avhengig av ulike tilstander. I python, kombinerer vi logiske operatorer, boolske verdier med `if` og `while` setninger for å styre utførelse av koden.

1.1 Boolsk verdier: er det sant eller usant?

En boolsk verdi er en datatype som bare kan være enten sann (`True`) eller falsk (`False`). Vi kan bruke boolske verdier til å teste om en betingelse er oppfylt. For eksempel kan vi sammenligne verdier for å sjekke om to tall er like eller om en verdi er større enn en annen.

[1]:
```
print("32 > 18:  ", 32 > 18)

alder = 32

# Opprett en boolsk variabel
voksen = alder >= 18
print(f"variabelen voksen,  verdien: {voksen},  type: {type(voksen)}")
```

```
32 > 18:   True
variabelen voksen,  verdien: True,  type: <class 'bool'>
```

1.2 Betingelser: styre utførelsen av kode

Med betingelser bruker vi boolsk verdier til å styre utførelsen av kode basert på om en bestemt betingelse er sann eller ikke.

1.2.1 `if`

Syntaks:

```
if betingelse:
    # kodeblokken med innrykk her. Den kjøres hvis betingelse er sann
    # [...]
```

[2]:
```
alder = 32

if alder >= 18:
    # Kodeblokken her kjøres hvis betingelsen er usann
    print("Du er en voksen")

print("Utenfor if-kodeblokken: alltid kjøres")
```

```
Du er en voksen
Utenfor if-kodeblokken: alltid kjøres
```

1.2.2 `if` med `else`

Vi kan legge til en kodeblokk som kjøres hvis betingelse var usann, med `else`:

Syntaks:

```
if betingelse:
    # Med innrykk. Kodeblokken kjøres hvis betingelse er sann
```

```
    # [...]
else:
    # Med innrykk. Den kjøres hvis betingelse er usann
    # [...]
```

Vi kan ha en `if` setning uten en `else` setning men en `else` setning trenger en `if` setning for å kunne defineres og `else` må være rett etter kodeblokken som er knyttet til `if`.

[3]:
```
alder = 16

if alder >= 18:
    # Kodeblokken her kjøres hvis betingelsen er usann
    print("Du er en voksen")
else:
    # Kodeblokken her kjøres hvis betingelsen var usann
    print("Du er et barn")
print("Utenfor kodeblokkene")
```

```
Du er et barn
Utenfor kodeblokkene
```

1.2.3 Flere tilfeller: `if`, `elif` og `else`

`elif` er en forkortelse for `else if`. Den brukes til å sjekke flere betingelser etter hverandre hvis den første `if`-betingelsen ikke er sann. Med `elif` kan vi legge til så mange tilfeller vi vil.

Syntaks:

```
if betingelse1:
    # Kodeblokken kjøres hvis betingelse1 er SANN
    # [...]
elif betingelse2:
    # Kodeblokken kjøres hvis:
    # betingelse1 er USANN og betingelse2 er SANN
    # [...]
else:
    # Kodeblokken kjøres hvis:
    # betingelse1 og betingelse2 er USANNE
    # [...]
```

[4]:
```
alder = 16

# En if-elif struktur
if alder >= 18:
    print("Du er en voksen")
elif alder > 12:
    print("Du er en tenåring")
```

```
Du er en tenåring
```

[5]:
```
alder = 4

# En if-elif-else struktur
if alder >= 18:
```

```
    print("Du er en voksen")
elif alder > 12:
    print("Du er en tenåring")
else:
    print("Du er et barn")
```

```
Du er et barn
```

[6]:
```
alder = 4

#  En if-elif-elif-else struktur
if alder >= 18:
    print("Du er en voksen")
elif alder > 12:
    print("Du er en tenåring")
elif alder < 5:
    print("Du er et småbarn")
else:
    print("Du er et barn")
```

```
Du er et småbarn
```

Merk at maksimalt én kodeblokk kjøres! (nøyaktig én hvis det også er en `else`-blokk)

[7]:
```
alder = 16

if alder >= 18:
    print("Du er en voksen")
elif alder > 5:
    print("Du er et barn")
    # Kodeblokken her aldri kjøres fordi:
    # hvis denne elif-betingelsen er sann, så var den elif betingelsen før!
elif alder > 12:
    print("Du er en tenåring")
else:
    print("Du er et småbarn")
```

```
Du er et barn
```

1.3 `While`-løkker: gjenta en kodeblokk mens en betingelse er sann

`while`-løkken brukes når vi vil fortsette å gjenta en kodeblokk så lenge en betingelse er sann.

Syntaks:

```
while betingelse:
    # kodeblokken med innrykk her
    # [...]
```

Vær forsiktig! Hvis betingelsen er aldri usann stopper ikke programmet! **Betingelse må oppdateres på en eller annen måte i løkken slik at det blir `False` etter en stund.**

Hvis du glemte å oppdatere betingelse og den blir aldri usann, kan du drepe et program ved å bruke `Ctrl-C` når programmet kjøres fra terminalen, ellers om du bruker en tolke kan de finnes en "stop" knapp for å avbryte

programmet.

I eksempelet nedenfor deler vi `1000` på 2 så lenge vi ikke har nådd `1` og vi skriver ut alle tallene innimellom.

[8]:
```
# Opprett variabelen før løkken
tall = 1000

# Fortsett å dele på 2 så lenge tallet er større enn 1
while tall > 1:

    # Oppdater variabelen knyttet til betingelsen
    # Del på 2
    tall = tall / 2
    print(tall)
```

```
500.0
250.0
125.0
62.5
31.25
15.625
7.8125
3.90625
1.953125
0.9765625
```

I eksempelet nedenfor fortsetter vi å be brukeren om å gi oss en streng til de skriver "Ha det".

Merk at i de to eksemplene her (i over- og undercellene) **må variabelen som brukes i betingelsen være definert før løkken starter**.

Merk også at avhengig av rekkefølgen på linjene i kodeblokken (først oppdater variabelen og deretter skriv ut, eller omvendt), kan enten den første eller den siste verdien ikke skrives ut.

- I eksemplen ovenfor **oppdaterer vi før** vi skriver ut, noe som betyr at den opprinnelige verdien (`1000`) aldri skrives ut.

- I eksemplen neden **oppdaterer vi etter** vi skriver ut, noe som betyr at den siste verdien (`Ha det`) aldri skrives ut.

[9]:
```
# Opprett variabelen før løkken
tekst = "Hei"
# Fortsett å be brukeren om å gi oss en streng så lenge de ikke skriver "Ha det"
while tekst != "Ha det":

    print(tekst)

    # Oppdater variabelen knyttet til betingelsen
    # Be brukeren om å skrive noe
    tekst = input("Skriv hva som helst. Skriv 'Ha det' for å slutte programmet")
```

```
Hei
Halloen
Jeg lærer python!
```