Modul 2

Hovedpoeng

- Semantisk og logiske feil er farlige! Vi "liker" kjøretidsfeil (syntaksisk feil) fordi programmet stopper umiddelbart og vi får en hjelpsom feilmelding. Med en logisk feil kjører dessverre programmet uten å stoppe, og vi legger kanskje ikke merke til at resultatet er feil!
- For å unngå store mengder semantiske/logiske feil, samtidig som vi forbedrer lesbarheten av koden vår, kan vi følge denne prosedyren:
    1. **Før vi begynner å skrive kode**, starter vi med å skrive beskrivelsen av hovedoppgavene (oppgaver på høyt nivå) vi ønsker å oppnå i programmet vårt med fulle norske setninger.
    2. Deretter bryter vi ned hovedoppgavene iterativt i flere mindre oppgaver (oppgaver på lavt nivå) til alle oppgavene på lavest nivå tilsvarer (nesten) én linje med kode.
    3. Bruk oppgavene på høyere nivå og oppgavene på lavere nivå som kommentarer i funksjonen.
    4. **Først nå kan du begynne å skrive kode**
- For å raskt oppdage feil i koden
    1. Tenk på flyten av en gitt inndata i koden din. Prøv med forskjellige inndata i tankene dine.
    2. Test koden din med noen gitte inndata (prøv å tenke på typiske inndata man ville bruke med denne koden, men også kanttilfeller som negative tall, 0, tomme lister eller strenger, osv.). Sjekk at resultatet er som forventet.
- Det er viktig å lese dokumentasjonen nøye for funksjonene/metodene vi bruker.
- Ikke vær redd for å søke informasjon på internett. Sørg bare for at du bruker riktige ord for å søke effektivt.
- Man trenger ikke å lære alt om alle de metode/bibliotekene som eksisterer, men i programmering er det viktig å lære å finne informasjon raskt om en gitt funksjon/bibliotek.
- Python har en rekke innebygde funksjoner som gir grunnleggende funksjonalitet og operasjoner
- Det er foranderlige sekvenstyper i Python (`list`) og uforanderlige sekvenstyper (`str`). I tillegg deler alle sekvenstyper felles metoder.
- En spesifikk type kan ha ekstra metoder. For eksempel lister og strenger.
- Husk hvilken type variablene du arbeider med har. Resultatet av en gitt operasjon kan avhenge av variabeltypen, og de tilgjengelige metodene avhenger også av variabeltypen.
- `for`-løkken brukes når vi vet antall ganger vi ønsker å gjenta koden på forhånd eller nå vi vet at vi vil iterere over en hel sekvens.
- `while`-løkken brukes når vi ikke kjenner antallet gjentakelser på forhånd, men vil fortsette å gjenta så lenge en betingelse er sann.

Jukseark

Påminnelse: det forventes ikke at dere kjenner det utenat! Men dere bør kunne finne informasjon raskt.

Datatyper

Man kan sjekke typen til et objekt med funksjonen `type()`.

Navn (norsk)	Navn (engelsk)	Eksempel	Konvertere til
Liste	List	`["Nora", "Odin", "Morten"]`	`list()`

Navn (norsk)	Navn (engelsk)	Eksempel	Konvertere til
Boolsk	Boolean	`True`, `False`, `12>0 and 12<28`	`bool()`

`None` er et spesielt objekt som representerer fraværet av en verdi eller en null-verdi. `None` er et unikt objekt av typen `NoneType`.

If-setninger

Maksimalt én kodeblokk kjøres! (hvis det er en 'else'-blokk, er det alltid nøyaktig én).

If

```
alder = 28
if alder >= 18:
    print("Du er en voksen")
```

If-else

```
alder = 28
if alder >= 18:
    print("Du er en voksen")
else:
    print("Du er en barn")
```

If-elif

```
alder = 28
if alder >= 18:
    print("Du er en voksen")
elif alder >= 13:
    print("Du er en tenåring")
```

If-elif-else

```
alder = 28
if alder >= 18:
    print("Du er en voksen")
elif alder >= 13:
    print("Du er en tenåring")
else:
    print("Du er et barn")
```

If-elif-elif-else

Det er ingen begrensning på hvor mange elif man kan ha i en if-setning.

```
alder = 28
if alder >= 18:
    print("Du er en voksen")
elif alder >= 13:
    print("Du er en tenåring")
elif alder < 1:
    print("Du er et spedbarn")
elif alder < 5:
    print("Du er et småbarn")
else:
    # 5 <= alder < 13
    print("Du er et barn")
```

Sammenligninger

Operator	Betydning
==	lik
!=	ikke lik
<	mindre enn
>	større enn
<=	mindre enn eller lik
>=	større enn eller lik
and	og
or	eller
not	ikke

Lister

```
tom_liste = []
land_liste = ["Norge", "Danmark", "Sverige"]
karakter_liste = [4, 3, 2, 8, 3]
rotete_liste = [2024, 0.1, "DIGI", True, None]

# Indeksen til det første elementet er `0`
print(land_liste[1])          # "Danmark"
```

Løkker

For-løkke med range

```
# Teller fra 0 til 9
# range(slutt)
for i in range(10):
    print(i)
```

```
# Teller fra 5 til 9
# range(start, slutt)
for i in range(5, 10):
    print(i)
```

```
# Teller fra 5 til 15 med steg på 2 (5,7,9,11,13,15)
# range(start, slutt, steg)
for i in range(5, 16, 2):
    print(i)
```

```
# Teller fra 10 til 6 med steg på -1 (10,9,8,7,6)
for i in range(10, 5, -1):
    print(i)
```

For-løkke og sekvenser

```
# Skriver ut landene i listen en etter en
liste = ["Norge", "Sverige", "Danmark"]
for land in liste:
    print(land)

# Skriver ut bokstavene og tegnene en etter en
streng = "Hallo, verden!"
for bokstav in streng:
    print(bokstav)
```

While-løkke

```
# Teller fra 0 til 5, og skriver ut "Etter løkken"
i = 0
while i <= 5:
    print(i)
    # Oppdater indeksvariabelen
```

```
    i = i + 1
print("Etter løkken")
```

Vanlige innebygde funksjoner og metoder

Innebygde funksjoner

Sekvens metoder

```
liste = [2024, 0.1, "DIGI", True, None]
streng = "Hei hei, verden!"
```

Indeksering syntaks: `liste[start:slutt:steg]` med standardverdier (dvs. om utelatt):

- `start = 0`
- `slutt = -1`
- `steg = 1`

Metode / Operasjon	Resultat	Beskrivelse
`liste[1]`	`0.1`	Hent ut det som ligger på indeks `1`
`streng[-1]`	`"!"`	Hent ut det som ligger `-1` fra slutten av sekvensen
`liste[-2]`	`True`	Hent ut det som ligger `-2` fra slutten av sekvensen
`streng[:3]`	`Hei`	Hent ut de som ligger fra indeksen 0 til (ikke inkludert) indeksen 3
`liste[1:4]`	`[0.1, 'DIGI', True]`	Hent ut de som ligger fra indeksen 1 til (ikke inkludert) indeksen 4
`streng[1:11:2]`	`'e e,v'`	Hent ut hver annen element, startende fra indeksen 1 til (ikke inkludert) indeksen 11
`len(liste)`	`5`	Lengden på sekvensen
`len(streng)`	`16`	Lengden på sekvensen
`["DIGI", True] + [2024, 0.1]`	`["DIGI", True, 2024, 0.1]`	Slå sammen sekvenser
`'verd' in streng`	`True`	Sjekk om noe er i sekvensen
`None not in liste`	`False`	Sjekk om noe ikke er i sekvensen
`liste.index("DIGI")`	`2`	Finn plassering til noe i sekvensen

Foranderlige sekvens metoder

Disse metodene er "in-place"-metoder. Det betyr at de ikke returnerer sekvensen, og i stedet oppdaterer de sekvensen på stedeten. Dette er mulig fordi vi her jobber med foranderlige sekvenser.

Metode / Operasjon	Resultat	Beskrivelse
`liste[1] = "!"`	`[2024, '!', 'DIGI', True, None]`	Sett det som ligger på indeks `1` til `"!"`
`liste.append(42)`	`[2024, '!', 'DIGI', True, None, 42]`	Legg noe til på slutten av sekvensen
`liste.insert(1, 42)`	`[2024, 42, '!', 'DIGI', True, None, 42]`	Sett noe inn i sekvensen på en gitt indeks
`liste.remove(42)`	`[2024, '!', 'DIGI', True, None, 42]`	Fjern et gitt element fra sekvensen.
`liste.pop(2)`	`[2024, '!', True, None, 42]`	Fjern elementet som ligger på en gitt indeks

Liste metoder

```
karakterene  = [1, 8, 3, 0, 10, 1]
vennene = ["Nora", "Odin", "Morten"]
```

Metode / Operasjon	Resultat	Beskrivelse
`karakterene.sort()`	`[0, 1, 1, 3, 8, 10]`	Sorter listen (om mulig!)
`vennene.sort(reverse=True)`	`['Odin', 'Nora', 'Morten']`	Sorter listen (om mulig!) motsatt vei

Streng metoder

Disse metodene er ikke "in-place"-metoder. De returnerer en ny sekvens og lar den opprinnelige sekvensen være intakt. Dette skyldes at vi her jobber med uforanderlige sekvenser.

Metode / Operasjon	Resultat	Beskrivelse
`streng.upper()`	`"HEI HEI, VERDEN!"`	Gjør alle bokstaver store
`streng.lower()`	`"hei hei, verden!"`	Gjør alle bokstaver små
`streng.startswith('hei')`	`False`	Sjekke om strengen begynner med
`streng.endswith('!')`	`True`	Sjekke om strengen slutter med
`streng.split(",")`	`['Hei hei', ' verden!']`	Deler strengen i en liste
`streng.split(" ")`	`['Hei', 'hei,', 'verden!']`	Deler strengen i en liste
`streng.replace('e', 'i')`	`Hii hii, virdin!`	Erstatter en understreng i strengen

Metode / Operasjon	Resultat	Beskrivelse
`streng.replace('ver', 'klo')`	`Hei hei, kloden!`	Erstatter en understreng i strengen
`" og ".join(vennene)`	`"Nora og Odin og Morten"`	Slår en list sammen til en streng