IN1000 - Uke 6

Skop og repetisjon

Plan for dagen

- Skop (nytt denne uken)
- Repetisjon av pensum hittil
  - En veldig kjapp oppsummering av input, typekonvertering og boolske uttrykk
  - Samlinger
  - Løkker
  - Prosedyrer, funksjoner og metoder
- Programmeringsstafett

Skop

lokal vs. global variabel

“Skop” vil si den delen av programmet du kan aksessere/få tilgang på en variabel. En variabel som er definert i en prosedyre eller funksjon kalles for en “lokal variabel”. Defineres en variabel utenfor en prosedyre eller funksjon vil den være en “global variabel” og dermed tilgjengelig for alle.

Generelt burde man bruke lokale variabler og heller lage funksjoner som returnerer verdier fremfor globale variabler som oppdateres av mange prosedyrer eller funksjoner.

Eksempel 1

```
def storst_av_to(tall1, tall2) :
    if tall1 > tall2 :
        return tall1
    return tall2

storst_av_to(3, 4)
```

Eksempel 2

```
def summer():
    print("sum: ", a + b)


a = 3
b = 2
summer()
```

Eksempel 2

```
def minus():
    a = 3
    b = 2


def summer():
    print("sum: ", a + b)


summer()
```

Repetisjon

Grunnleggende: input() og typekonvertering

- Vi tar input fra bruker med funksjonen **input()**
- **input()** returnerer en **streng**
- For å skrive ut tekst til brukeren skriver vi **input(“Tekststreng”)**
- For å konvertere bruker vi:
  - **int()** for å konvertere til heltall
  - **float()** for å konvertere til flyttall

```
1   # Eksempler:
2
3   # Ta inn streng fra bruker:
4   streng = input("Skriv inn en streng: ")
5
6   # Ta inn heltall fra bruker:
7   heltall = int(input("Skriv inn et heltall: "))
8
```

Grunnleggende: Boolske uttrykk

- Et **boolsk uttrykk** er et uttrykk som evaluerer til **True** eller **False**
- Vi kan sette sammen flere boolske uttrykk med **and, or** og **not**
- Vi kan sammenligne verdier med operatorene ==, **!**=, >, <, >= og <=.

```
13   ## Boolske uttrykk
14
15   a and b      # True hvis og bare hvis både a og b er True
16   a or b       # True hvis og bare hvis enten a eller b er True
17   not a        # True hvis og bare hvis a er False
18
19   10 == 10     # True
20   10 != 10     # False
21   10 > 10      # False
22   10 >= 10     # True
```

Samlinger

- Tre viktige typer samlinger: **lister, ordbøker og mengder**.
- En **liste** inneholder et vilkårlig antall verdier i en fast rekkefølge, der vi kan ha mange av samme verdi.
- En **mengde** har ingen rekkefølge og vi kan bare ha én av hver verdi.
- En **ordbok** består av en mengde med **nøkler** som hver er knyttet til en **verdi**.
    - Siden nøklene er en mengde, kan vi ikke ha to like nøkler - men vi kan ha like verdier.

Eksempel: Liste

```
29    liste = [1, 2, 3, 4, 5] # Lager en ny liste med elementene 1, 2, 3, 4 og 5
30    liste.append(6)         # Legger til tallet 6 i listen
31    print(liste[0])         # Skriver ut det første elementet i listen
32                            # (elementet på indeks 0)
33    liste[3] = 0            # Setter verdien av elementet på indeks 3 til 0
34    liste.insert(3, 5)      # Setter inn verdien 5 på indeks 3 og forskyver
35                            # resten av listen bakover (sletter ingen elementer)
36    liste.pop(0)            # Fjerner elementet på indeks 0 og returnerer det
37    liste.remove(5)         # Finner verdien 5 i listen og fjerner den
38    len(liste)              # Returnerer antall elementer i listen
```

Eksempel: Mengde

```
41  ## Mengder:
42
43  mengde = set()          # Lager en tom mengde
44  mengde = set(liste)     # Gjør om liste til en mengde (fjerner duplikater)
45  mengde.add(10)          # Legger til 10 i mengden
46  mengde.remove(10)       # Fjerner 10 fra mengden (gir feilmelding dersom 10 ikke er i mengden)
47  mengde.discard(10)      # Fjerner 10 fra mengden, men gir ingen feilmelding
48  len(mengde)             # Returnerer antall elementer i listen
```

Eksempel: Ordbok

```
51  ## Ordbøker
52
53  # Definerer ordbok med tre verdier:
54  ordbok = {"Nøkkel1" : "Verdi1",
55            "Nøkkel2" : "Verdi2",
56            3 : 3}
57  ordbok["Nøkkel1"]           # Returnerer verdien til nøkkel1
58  ordbok[3] = 2               # Setter verdien av ordbok[3] til 2
59  ordbok["Nøkkel3"] = "Verdi3"# Oppretter ny nøkkel 4 og setter verdi til 4
60  ordbok.pop("Nøkkel3")       # Fjerner nøkkel3 og verdien fra ordboken
61  len(ordbok)                 # Returnerer antall nøkler i ordboken
```

Løkker

for-løkker

- Itererer over en *samling*
- range(10) er for eksempel også en samling!

while-løkker

- forutsetter at uttrykket etter **while** er sant for å kjøre koden i kodeblokken
- avsluttes når uttrykket er usant
- pass på evige løkker - uttrykket burde kunne evaluere til usant

Prosedyrer, funksjoner og metoder

	Prosedyrer	**Funksjoner**	**Metoder**
Tar argumenter	0 eller flere	0 eller flere	0 eller flere
Defineres med `def`	Ja	Ja	Ja
Har returverdi	Nei	Ja	Noen ganger
Hører til en klasse	Nei	Nei	Ja

Prosedyrer og funksjoner

Prosedyre

- Har ikke en returverdi

Funksjon

- Har en returverdi
- Denne lagres vanligvis i en variabel
- Funksjoner returnerer kun én ting
- Når funksjonen returnerer så avsluttes den

Programmeringsstafett

Programmeringsstafett

Sett dere i team på tre og tre.

Én person i teamet skal programmere, men får ikke lov til å kommunisere med de to andre. De to andre forteller den som skal programmere, hva personen skal skrive.

Oppgave 1

Lag en funksjon som tar inn en streng og reverserer strengen før den returneres.

Oppgave 2

Skriv en funksjon som tar inn en streng og sjekker om strengen er et pangram.

Et pangram er en setning som inneholder alle bokstavene i alfabetet. Eksempel: Jeg begynte å fortære en sandwich mens jeg kjørte taxi på vei til quiz.

Funksjonen skal returnere True hvis strengen er et pangram, False ellers.

Oppgave 3

Lag en funksjon som sjekker om to tall er ved siden av hverandre på en tallrekke med heltall. Returner True om det stemmer.

Oppgave 4

Lag en funksjon som sjekker om tallene i en liste parvis er ved siden av hverandre på en tallrekke med heltall/”nabotall”. Returner hvor mange par som er nabotall i listen. (Bruk løsningen fra forrige oppgave)

Eksempel:

```
nabotall([1, 2, 4, 6, 4, 3, 9])          # Skal returnere 2
```

Første par er 1 og 2 og tallene er nabotall; nabotalll = 1

Det andre paret er 4 og 6, og er ikke nabotall..

Det tredje paret er 4 og 3, de er nabotall. naboantall = 2

Det siste paret, 3 og 9, er ikke nabotall.

Oppgave 5

Lag en prosedyre som tar inn et tall, og skriver ut tallet i streng-format. 1 blir “en”, 19 blir “en ni”.

Oppgave 6

Lag en funksjon kan_deles_på som sjekker om et tall kan deles på to argumenter.

Eksempler

```
kan_deles_på(-12, 2, 6)  -> true    # fordi -12 kan deles på 2 og 6
kan_deles_på(-12, 2, -5) -> false  # fordi -12 ikke kan deles på 5
kan_deles_på(45, 1, 6)   -> false  # fordi 45 ikke kan deles på 6
kan_deles_på(45, 5, 15) -> true   # fordi 45 kan deles på 5 og 15
kan_deles_på(4, 1, 4)     -> true   # fordi 4 kan deles på 1 og 4
kan_deles_på(15, -5, 3)  -> true   # fordi 15 kan deles på -5 og 3
```

Oppgave 7

Hvis vi lister opp alle naturlige tall under 10 som er med i 3- eller 5-gangen så får vi 3, 5, 6 og 9. Summen av disse er 23.

Lag en funksjon som returnerer summen av alle tall som er med i 3- eller 5- gangen som også er *mindre* enn tallet som blir sendt som argument. I tillegg, hvis tallet er negativt, returner 0.

Obs: Hvis tallet er med i *både* 3- og 5-gangen (eks. 15), tell det med kun én gang.

Oppgave 8 (M)

Gitt en liste med tall, finn det ene tallet som dukker opp et oddetall antall ganger.

Dere kan anta at det alltid vil være ett tall som dukker opp et oddetall antall ganger.

Eksempler

[7] burde returnere 7, fordi det kun dukker opp 1 gang (1 er et oddetall).

[0] burde returnere 0, fordi det kun dukker opp 1 gang (1 er et oddetall).

[1, 1, 2] burde returnere 2, fordi det dukker opp 1 gang (1 er et oddetall).

[0, 1, 0, 1, 0] burde returnere 0, fordi det dukker opp 3 ganger (3 er et oddetall).

[1, 2, 2, 3, 3, 3, 4, 3, 3, 3, 2, 2, 1] burde returnere 4, fordi det dukker opp 1 gang (1 er et oddetall).