REPETISJONSKURS

Variabler, konstanter, blokker og skop

Lars Ivar og Magnus

VARIABLER

- Et bestemt navn som representerer en verdi.

VARIABLER

- Et bestemt navn som representerer en verdi.

```
foo = "En streng verdi"
```

VARIABLER

- Et bestemt navn som representerer en verdi.

```
foo = "En streng verdi"
```

VARIABLER

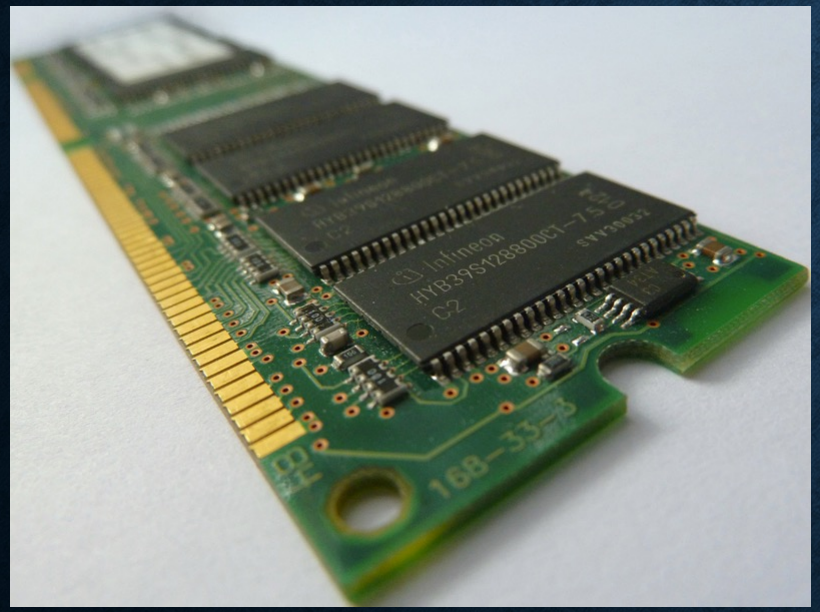
- Et bestemt navn som representerer en verdi.

```
foo = "En streng verdi"
```

- Vi kan si at vi tilordner en verdi til en variabel
- Vi kan også si at variabelen peker på strengen 'En streng verdi'
- Vi kan endre verdien ved å si «foo = ''ny verdi''»

VARIABLER

- Ulike metaforer for variabler



HVORFOR BRUKE VARIABLER?

- Kan ta vare på en verdi og bruke den senere i programmet.
- Gjør koden mer oversiktlig.
  - Lett å forstå:

```
lengde = 5
hoyde = 2
areal = lengde * hoyde
```

VARIABELNAVN

- Bør ha gode variabelnavn som forklarer hvilken verdi variabelen skal ha.
- Variabelnavnet bør ikke være likt som andre varibelnavn i programmet.
- Typiske dårlige variabelnavn kan være: tall, bokstav, liste, list, int, str, streng, ordbok…

KONSTANTER

- En konstant er en variabel med uforanderlig verdi.
- I fysikken så har man blant annet konstanten av lysets hastighet som er: 299792458 m/s
- Python har ikke ekte konstanter, men..
- Vi signaliserer at vi lager en konstant ved å kun ha store bokstaver i variabelnavnet. Dette sier til andre som leser koden at disse variablene/konstantene ikke skal forandres.

```
ANTALL_RADER = 10
FARGE_CELLER = "#ffffff"
```

DATA TYPER

- En variabel kan ha mange ulike typer verdi. Vi har:
  - Primitive datatyper
    - Heltall
    - Flyttall
    - Boolske verdier
    - Strenger

```
heltall = 5
flyttall = 5.1
boolsk = True
boolsk2 = False
streng = "Hei"
streng2 = 'Hei'
```

- Kan sjekke dette med funksjonen type(arg) som returnerer typen til en verdi.

```
print(type(streng2)) # -> <class 'str'>
```

FLERE DATATYPER

- Vi har også andre innbygde datatyper som:
  - liste / list
  - Ordbok / dict
  - Mengde / set

```
liste = [3, 1, 2]
ordbok = {"a": 1, "b": 2}
mengde = {1, 2, 3}
```

- (ikke så relevant for dette repkurset) Vi kan også lage egne datatyper ved å lage klasser. Et objekt av en klasse har klassen som datatype.

```
print(type(Sang())) # -> <class '__main__.Sang'>
```

KORT OM UTTRYKK

- Et uttrykk er en sammensetting av **verdier** og operatorer eller funksjoner som evalueres til en verdi.
- Starter alltid med det innerste uttrykket og jobber oss utover.
- Eks:

```
g = (4 * (3 * (2 - 1)))
```

- Her starter vi med å evaluere uttrykket (2 – 1), så (3 * 1) og til slutt (4 * 3)
- g får verdien 12

FORSKJELLIGE UTTRYKK

- Aritmetiske uttrykk:
  - *, /, +, -, %, //, **
  - 4 * (3 / 2)
- Boolske uttrykk:
  - ==, >, <, and og or
  - True and 3 == 2 → False

False

BLOKKER

```
def funksjon():
    heltall = 12
    if (heltall == 12):
        tekst = "Tallet er 12!"

class Hund:
    def __init__(self, navn):
        self._navn = navn
        self._alder = 0

    def haBursdag(self):
        for i in range(3):
            print("Bjeff")
        self._alder += 1
```

- I python vil en blokk være den koden som er på samme **indenteringsnivå.**
- Vi har blokker etter løkker, if-uttrykk, funksjoner, klasser osv.
- En blokk definerer et **skop.**

SKOP

- Hvilke variabler som man har tilgang til i en blokk.
- Variabelen må være tilgjengelig i skopet man er i for at man skal kunne bruke den.
- Globale variabler – variabler som er tilgjengelige for hele programmet (gitt at ingen lokale variabler overskygger den)
- Lokale variabler – variabler opprettet i en funksjon eller klasse som ikke er tilgjengelig utenfor (skopet til) funksjonen.

SKOP

- Parametere er alltid lokale variabler i prosedyren eller funksjonen sin.
- Dårlig kodeskikk å bruke globale variabler i python, vi prøver å holde oss til lokale variabler

```
g = "hei"
#g er en global variabel

def funk(a):
    #a er en lokal variabel i funksjonen funk
    b = 4
    #b er en lokal variabel i funksjonen funk

    print(g)

    return a * b

funk(3)
```

SKOP

- Dersom en variabel finnes både lokalt og globalt overskriver den lokale (eller mest lokale) variabelen den globale.
- Eks:

```
g = "hei"

def pros():
    g = "hade"

    print(g)
    #skriver ut: hade

pros(3)
```