OBJEKTORIENTERT PROGRAMMERING

KLASSER, OBJEKTER OG REFERANSER

MIAU@IFI.UIO.NO

Plan for dagen

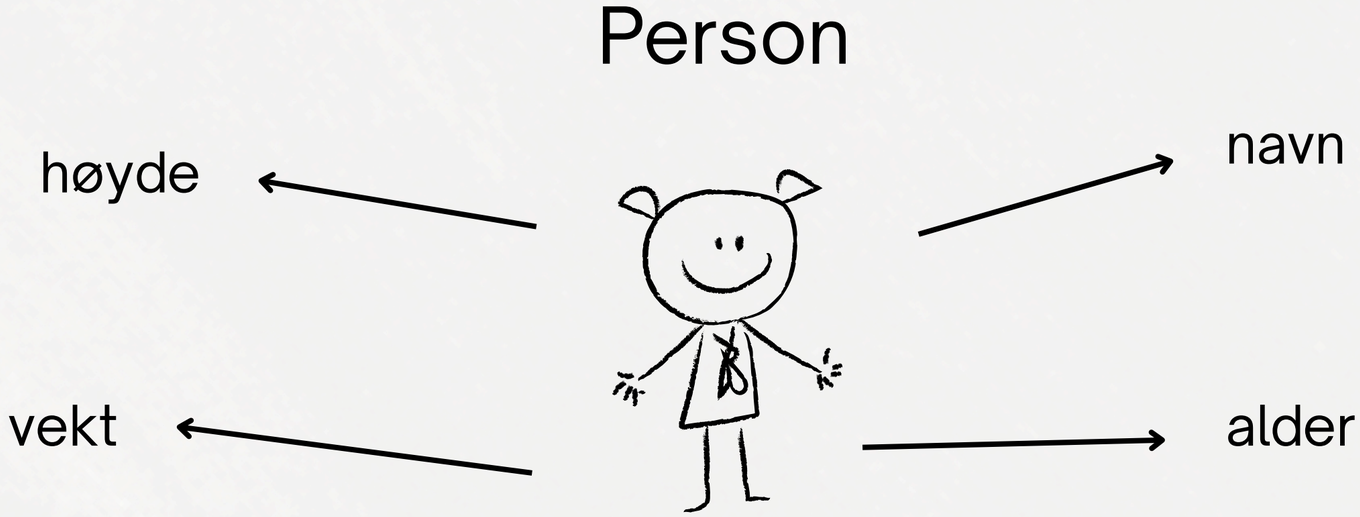
- Klasser og objekter
- Syntaks
- Metoder
- Hvordan bruker man en klasse?
- Grensesnitt
- Innkapsling
- Referanser
- Magiske metoder

Klasser og objekter

- En klasse er en slags mal/oppskrift for noe
  - Forklarer hvordan et objekt skal oppføre seg og hvordan det skal se ut
- Et objekt er en konkret ting laget fra denne oppskriften
- Tenk på en klasse som en gruppe, og et objekt som et medlem av denne gruppen

Klasser og objekter

Klassen Person:



Egenskaper alle objekter av en klasse har (instansvariabler)

Eksempel

Karoline

høyde = 167

navn =
‘Karoline’



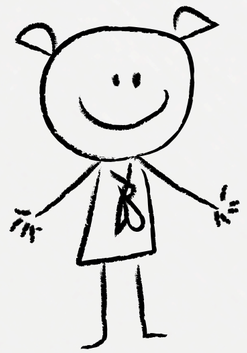
vekt = 62

alder = 19

Justas

høyde = 190

navn =
‘Justas’



vekt = 70

alder = 20

Generell syntaks

```
class Klasse:
    def __init__(self, parameter1, parameter2):
        self._egenskap1 = parameter1
        self._egenskap2 = parameter2
```

Eksempel på Person-klasse

```
class Person:
    def __init__(self, navn, alder, høyde, vekt):
        self._navn = navn
        self._alder = alder
        self._høyde = høyde
        self._vekt = vekt
```

Klassenavn

Instansvariabler

Konstruktør

Metoder

- Funksjoner og prosedyrer som er hører til en klasse.

```
class Klasse:
    # konstruktør med instansvariabler

    def metode1(self, parameter1, parameter2):
        # gjør noe

    def metode2(self, paramter1, parameter2):
        # gjør noe
```

Hvordan bruker man en klasse?

Du må laste inn klassen fra en fil, til en testfil:

```
# from <filnavn> import <Klasse>
from person import Person
```

Oppretter et objekt av klassen:

```
anne = Person('Anne', 25, 160, 58)
```

Kaller på en metode, på objektet:

```
anne.endre_navn('Dorte')
```

Grensesnitt

- Beskriver hva en klasse skal kunne gjøre
  - Hvilke metoder den skal ha
  - Men, ikke hvordan de er implementert
- Et slags skjelett

Innkapsling

- Går ut på å skjule detaljer og beskytte data inni et objekt, slik at de bare kan brukes på en kontrollert måte.
- Ofte gjør vi dette ved å bruke private variabler (starter med __ eller _) og metoder som gir trygg tilgang

```
class Konto:
    def __init__(self, saldo):
        self.__saldo = saldo  # "privat" variabel

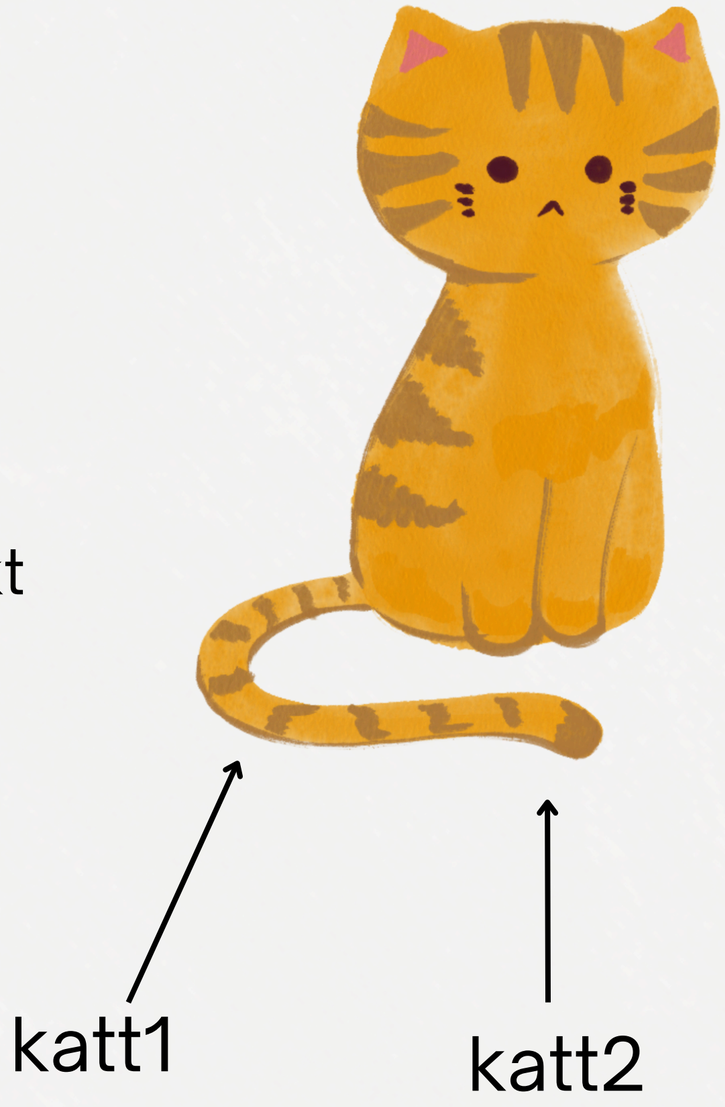
    def hent_saldo(self):
        return self.__saldo
```

Referanser

- Vi bruker referanser for å få tak i objekter
- En referanse er noe (f.eks. et navn) som peker på/henviser til et objekt i minnet
    - Den *er* ikke objektet, men en *peker* til objektet

Referanser

- Vi bruker referanser for å få tak i objekter
- En referanse er noe (f.eks. et navn) som peker på/henviser til et objekt i minnet
  - Den *er* ikke objektet, men en *peker* til objektet
  - Flere referanser kan til referere til samme objekt
    - katt1 = katt2
  - Om man endrer på det referte objektet (vha. en metode) med et av navnene, endres det for alle
    - katt2.endre_farge(“svart”)



Referanser

- Vi bruker referanser for å få tak i objekter
- En referanse er noe (f.eks. et navn) som peker på/henviser til et objekt i minnet
  - Den *er* ikke objektet, men en *peker* til objektet
  - Flere referanser kan til referere til samme objekt
    - katt1 = katt2
  - Om man endrer på det referte objektet (vha. en metode) med et av navnene, endres det for alle
    - katt2.endre_farge(“svart”)



Referanser

Hva blir resultatet av å kjøre denne koden?

```
a=1
b=2
c=3
listeA = [a,b,c]
listeB = [1,2,3]
listeC = listeA
listeA.append(5)
listeC.append(6)
print(listeC[4])
```

- ○ 4 skrives ut
- ○ 5 skrives ut
- ○ IndexError: list index out of range
- ○ 6 skrives ut

Referanser

Hva blir resultatet av å kjøre denne koden?

```
a=1
b=2
c=3
listeA = [a,b,c]
listeB = [1,2,3]
listeC = listeA
listeA.append(5)
listeC.append(6)
print(listeC[4])
```

- ○ 4 skrives ut
- ○ 5 skrives ut
- ○ IndexError: list index out of range
- ● 6 skrives ut

Magiske metoder

- **__str__** bestemmer hva som skjer når vi skriver *print(objekt)*
- **__eq__** bestemmer hva som skjer når vi sammenligner to objeker med hverandre vha. ==
- **__init__** bestemmer hva som skjer når vi oppretter et objekt

__str__

Bestemmer hva som skrives til terminalen når man skriver *print(objekt)*

```
class Genser:
    def __init__(self, merke, størrelse):
        self._merke = merke
        self._størrelse = størrelse

    def __str__(self):
        return f'Genser fra {self._merke} i størrelsen {self._størrelse}'

genser = Genser('Zara', 'S')
print(genser)
```

Vil skrive ut “Genser fra Zara i størrelsen S”

__eq__

- Om vi ikke definerer **__eq__**-metoden fungerer ‘==’ på samme måte som ‘is’

```
stue == soverom
stue is soverom
```

- Begge sjekker om referansene peker til det samme objektet i minnet

__eq__

__eq__ bestemmer hva som skal returners når man skriver objekt1 == objekt2

```
class Rom:
    def __init__(self, bredde, lengde):
        self._bredde = bredde
        self._lengde = lengde

    def hent_lengde(self):
        return self._lengde

    def hent_bredde(self):
        return self._bredde

    def __eq__(self, annet_rom):
        return (self._lengde == annet_rom.hent_lengde() and self._bredde == annet_rom.hent_bredde())

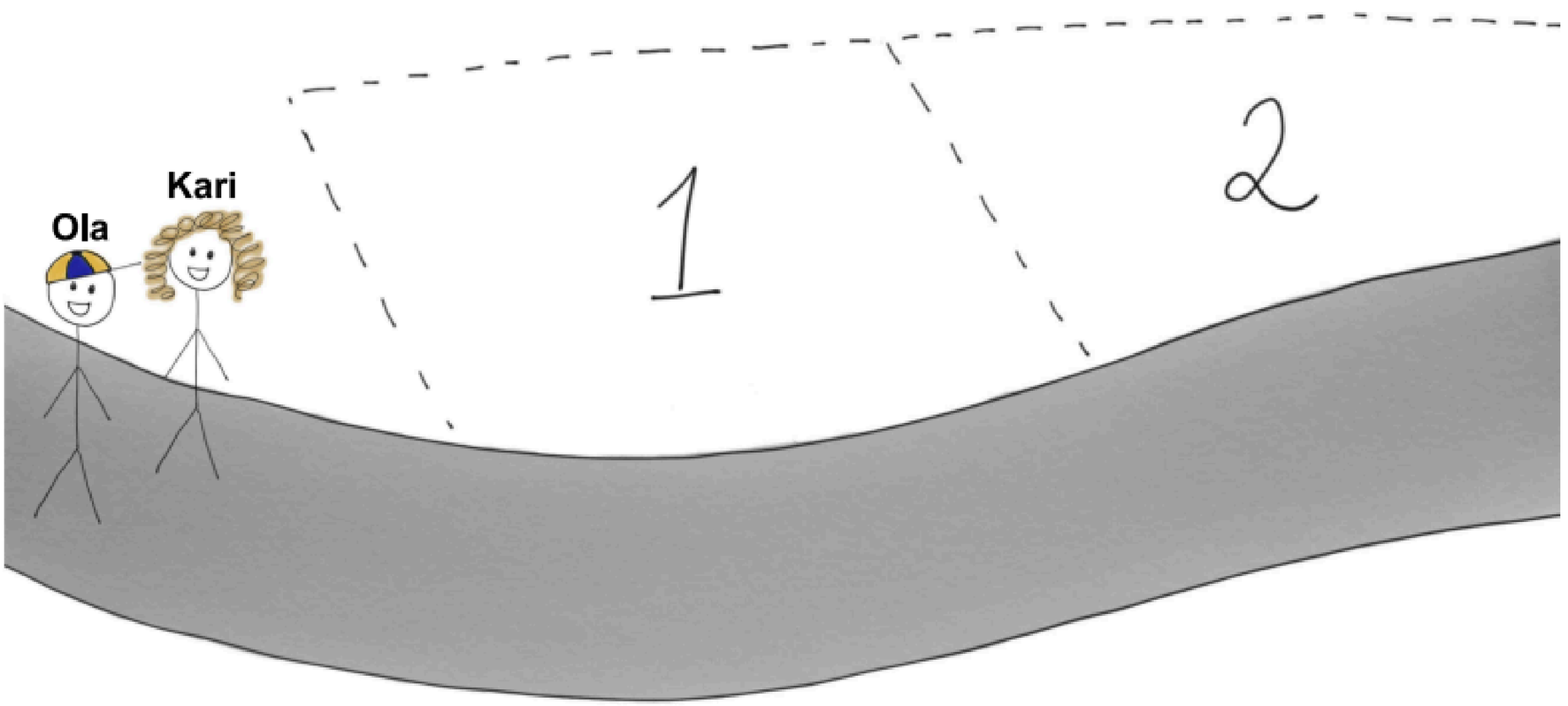

stue = Rom(10,5)
soverom = Rom(10,5)

print(stue == soverom)
```

Vil skrive ut True da bredden og lengden på de to objektene er like.

Minnegaten

Ledige tomter = Ledig minne



karis_hus = Hus("blå")

Først bygges det et blått hus på en ledig tomt



karis_hus = ~~Hus("blå")~~ 1



karis_hus = ~~Hus("blå")~~

olas_hus = Hus("rød")



karis_hus = ~~Hus("blå")~~

olas_hus = ~~Hus("rød")~~



karis_hus = ~~Hus("blå")~~

olas_hus = ~~Hus("rød")~~

olas_hus = karis_hus



karis_hus = ~~Hus("blå")~~ 1

olas_hus = ~~Hus("rød")~~ 2

olas_hus = karis_hus 1

Ola sitt hus er det samme som Kari sitt hus



olas_hus.mal("gul")



olas_hus.mal("gul")



```
olas_hus.mal("gul")

karis_hus.hent_farge()
```

Hvilken farge har Kari sitt hus nå?



```
olas_hus.mal("gul")

karis_hus.hent_farge()

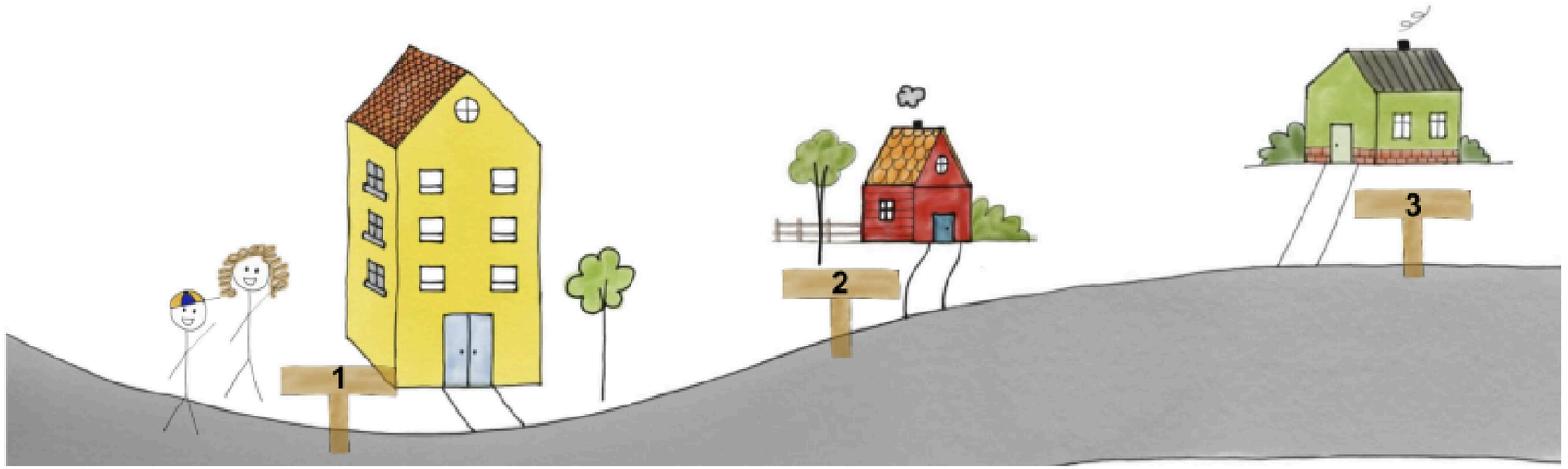
olas_hus = Hus("grønn")
```



```
olas_hus.mal("gul")

karis_hus.hent_farge()

olas_hus = Hus("grønn")
```

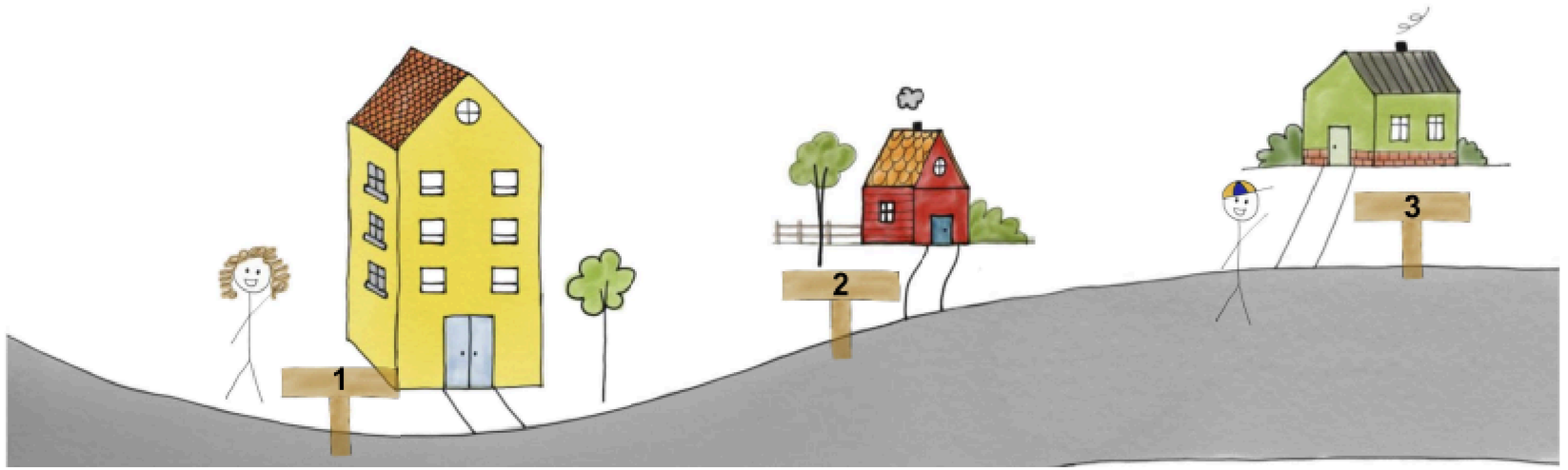


```
olas_hus.mal("gul")

karis_hus.hent_farge()

olas_hus = ~~Hus("grønn")~~ 3
```

Ola sitt hus er endret til å være det nye huset



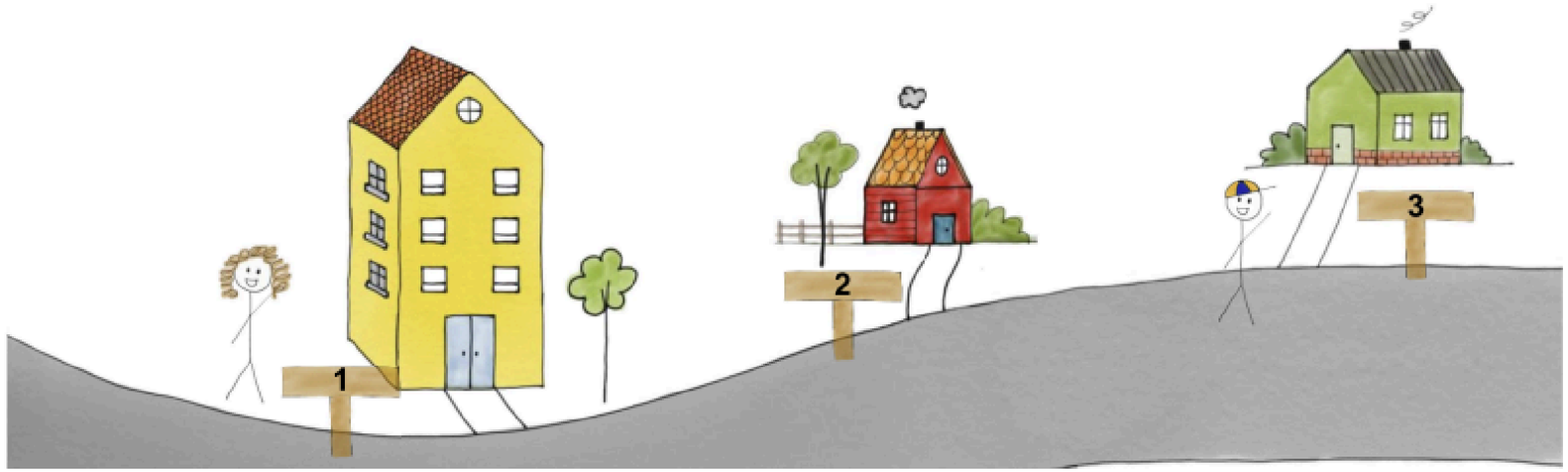
```
olas_hus.mal("gul")

karis_hus.hent_farge()

olas_hus = ~~Hus("grønn")~~ 3

karis_hus.hent_farge()
```

Hvilken farge har Kari sitt hus nå?

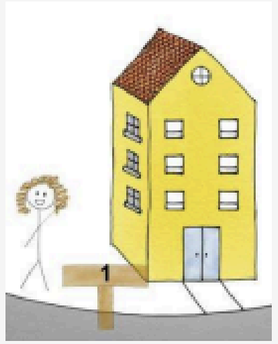


referanser (minne-adresser) kan vi i utgangspunktet se ved å printe objektene

```
print(olas_hus)
print(karis_hus)
```

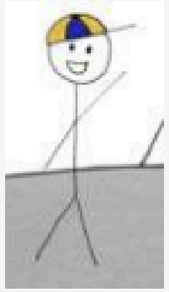



```
<__main__.Hus object at 0x00000214D83EFEE0>
<__main__.Hus object at 0x00000214D83EFA90>
```



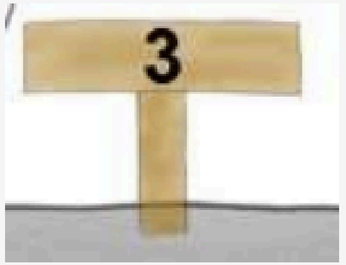
variabel `olas_hus`

(merkelapp vi setter på en verdi)



referanse `0x00000214D83EFA90`

(hvor verdien ligger i minnet)



objekt `__main__.Hus object`

(selve verdien)

