GRUPPETIME
UKE 7

IN1000 GRUPPE 25 - 11.10.21

PLAN FOR GRUPPETIMEN

- Oblig 5 - hvordan gikk det?
- Jobbe sammen på MetroRetro
- OOP: objektorientert programmering
- Klasser og objekter
- Innkapsling og grensesnitt

LÆRDOMMER FRA OBLIG 5

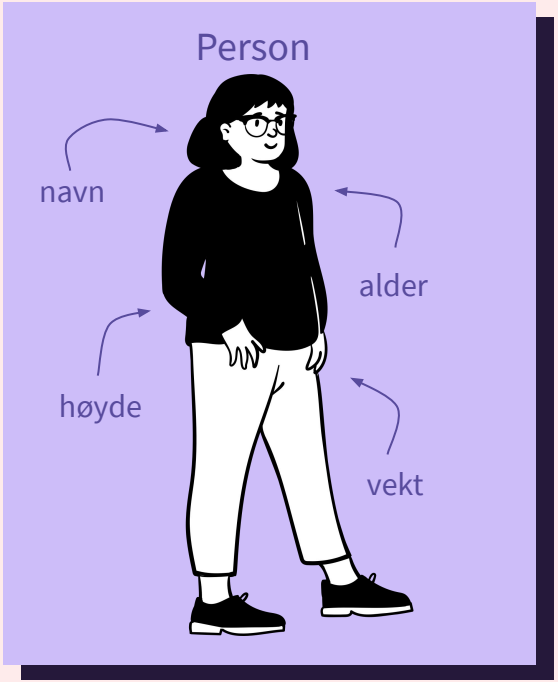
- Ha gjerne med en assert statement
  ```
  assert street_number > 0, "Street number must be above 0"
  ```
- Returner en variabel, ikke returner beregninger direkte
  - Unngå: `return a + b`
- Oppgave 2 - unngå while True, skriv heller:
  ```
  svar = ""
      while svar != "ja":
          print("Programmet kjører så lenge du vil")
          svar = input("Vil du avslutte programmet? ").lower()
  ```
- Oppgave 4 - inkluder **ALLE** nødvendige filer for at programmet skal kjøres
- Skriv enten camelCase eller snake_case konsistent
  - PEP8 bruker snake_case
- ALDRI navngi variabler, funksjoner etc. med æ, ø eller å

LÆRINGSMÅL UKE 7

- Kjenne til motivasjon og bakgrunn for objektorientert programmering
- Kunne definere en klasse med instansvariabler, metoder og konstruktør
- Kunne opprette objekter av egendefinert klasse og bruke deres tjenester gjennom metodekall
- Forstå sentrale begreper som grensesnitt og innkapsling
- Kjenne til utviklingsprosessen for en klasse gjennom design, implementasjon og testing

OOP: Hva er OOP?

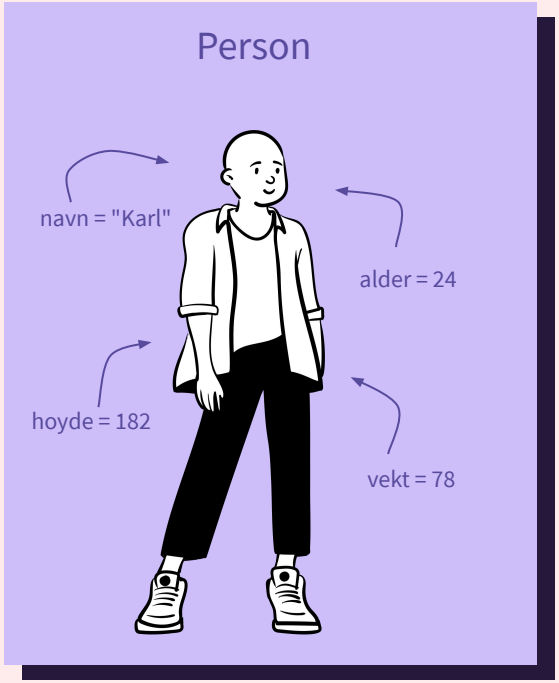
Hva er felles egenskaper for en Person?

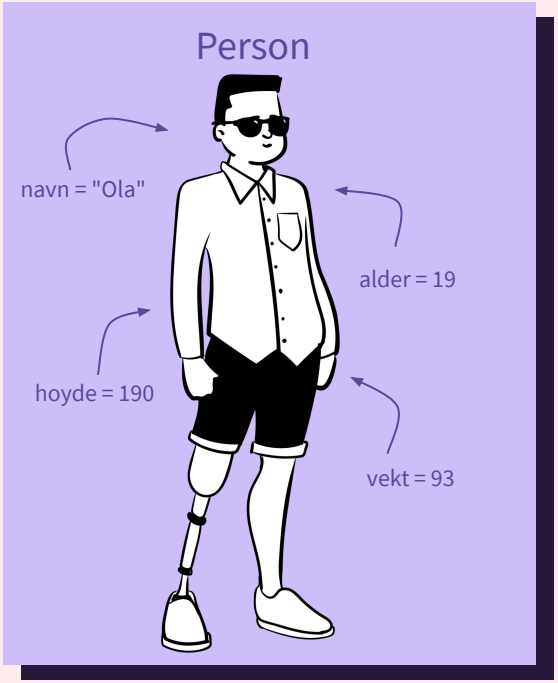


OOP
OBJEKTORIENTERT PROGRAMMERING

- Handler om å “modellere” verden → forenkle virkeligheten
- Hvordan kan vi forenkle virkeligheten?
  - Gruppere → finne felles egenskaper til en gruppe
- Tenk på en klasse som en gruppe, og et objekt som et medlem av denne gruppen
- Klasse (gruppe)
  - Instansvariabler
    felles egenskaper alle objekter av en klasse har
  - Metoder
    handlinger vi kan gjøre med objektene
    skaper innkapsling og brukergrensesnitt

OOP: lage objekter av en klasse





METRORETRO OPPGAVE 1

person.py

person.py

Man kan skrive en klasse inn i hvilken som helst fil, men i dette kurset ønsker vi én fil per klasse!

```python
class Person:
    def __init__(self, navn, alder, vekt, hoyde):
        self._navn = navn
        self._alder = alder
        self._vekt = vekt
        self._hoyde = hoyde

    def get_navn(self):
        return self._navn

    def set_navn(self, nytt_navn):
        self._navn = nytt_navn

    def skriv_ut_hilsen(self):
        print("Hei, jeg heter",  self._navn, "og jeg er",
              self._alder, "aar gammel")

    def hoyere_enn(self, annen_person):
        if self._hoyde > annen_person._hoyde:
            return True
        return False
```

Kunne ha laget en getter metode for høyde

METRORETRO OPPGAVE 2

test_person.py

test_person.py

Vi oppretter objekter og tester metodene våre i en separert test fil. Derfor må vi importere klassen fra en fil.
Syntaks:
from filnavn import Klasse

```python
from person import Person

silje = Person("Silje", 12, 46, 151)
per = Person("Per", 68, 84, 189)
ole = Person("Ole", 25, 76, 191)

silje.skriv_ut_hilsen()
per.skriv_ut_hilsen()
ole.skriv_ut_hilsen()

person_liste = [silje, per, ole]
for person in person_liste:
      person.skriv_ut_hilsen()

# Vi lager objekter basert på Karl og Ola som vi så på tidligere slides
karl = Person("Karl", 24, 78, 182)
ola = Person("Ola", 19, 93, 190)

karl.skriv_ut_hilsen()
karl.set_navn("Sara")
karl.skriv_ut_hilsen()

def hent_hoyeste_person(personer):
    hoyest = personer[0]
    for i in range(1, len(personer)):
        if personer[i].hoyere_enn(hoyest):
            hoyest = personer[i]
    return hoyest

print("Høyest:", hent_hoyeste_person(person_liste).)
```

Lik utskrift

Fordeler og ulemper med setter metode?

klassenavn

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

    def get_navn(self):
        return self._navn

    def set_navn(self, nytt_navn):
        self._navn = nytt_navn

    def skriv_ut_hilsen(self):
        print("Hei, jeg heter",  self._navn, "og jeg er",
              self._alder, "aar gammel")

    def hoyere_enn(self, annen_person):
        if self._hoyde > annen_person._hoyde:
            return True
        return False
```

instansvariabler

konstruktør

metode

funksjons-metode

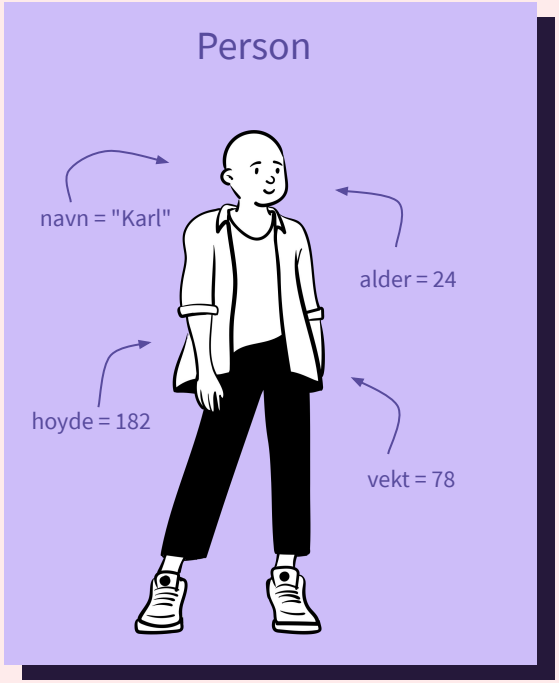
```
karl = Person("Karl", 24, 78, 182)
ola = Person("Ola", 19, 93, 190)
```

vi lager objekter av klassen Person

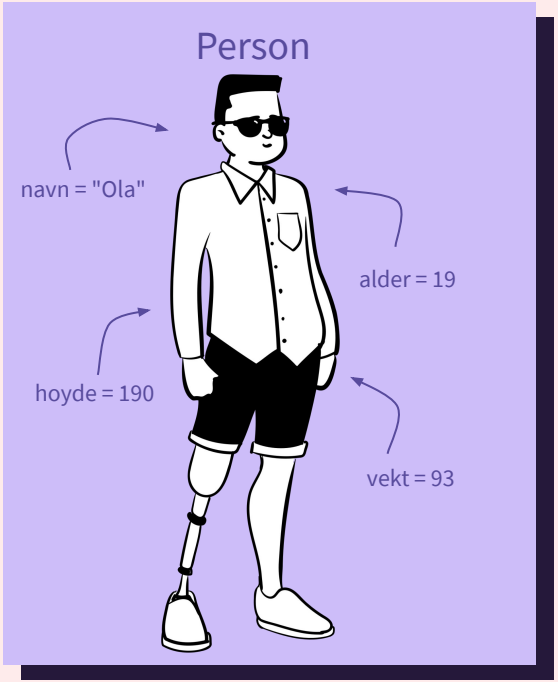
```
print(ola.hoyere_enn(karl))
print(karl.hoyere_enn(ola))
```

metodekall på objektene karl og ola

OOP: lage objekter av en klasse




```
karl = Person("Karl", 24, 182, 78)
```




```
ola = Person("Ola", 19, 190, 93)
```

OOP: metoder prosedyrer/funksjoner på objekter

INNKAPSLING & GRENSESNITT

- Alle klasser har et public (tilgjengelig utenfra) interface, et **grensesnitt**, altså hvilke metoder du kan bruke og hva metodene gjør.

- Eks: du har et objekt “bil” den har en metode som heter “kjor(km)”
  - Vi vet ikke helt hvordan den er definert, vi vet bare at den tar bilen på en kjøretur i oppgitt antall km, og kanskje da oppdaterer tanken og kilometerstanden (fordi den har kjørt)
  - Altså, vi vet ikke hvordan det gjøres, vi bare vet at det skjer når vi kaller metoden, dette kalles **innkapsling**.
    - Man kan kjøre en bil ved hjelp av rattet og pedalene uten å vite hvordan motoren fungerer.
      Slik er det også med objekter → kan bruke metoder uten å vite hvordan de er implementert.

Nysgerrig på de fire prinsippene i OOP?

```python
class Person:
    def __init__(self, navn, alder, vekt, hoyde):
        self._navn = navn
        self._alder = alder
        self._vekt = vekt
        self._hoyde = hoyde

    def get_navn(self):
        return self._navn

    def set_navn(self, nytt_navn):
        self._navn = nytt_navn

    def skriv_ut_hilsen(self):
        print("Hei, jeg heter",  self._navn, "og jeg er",
              self._alder, "aar gammel")

    def hoyere_enn(self, annen_person):
        if self._hoyde > annen_person._hoyde:
            return True
        return False
```

implementasjon
person.py

```python
karl = Person("Karl", 24, 78, 182)
ola = Person("Ola", 19, 93, 190)

karl.skriv_ut_hilsen()
print(ola.hoyere_enn(karl))
print(karl.hoyere_enn(ola))
```

grensesnitt
test_person.py

METRORETRO OPPGAVE 3

dyr.py og test_dyr.py

```python
class Dyr:
    def __init__(self, art, kjonn, vekt):
        self._art = navn
        self._kjonn = alder
        self._vekt = vekt

    def skriv_ut_hilsen(self):
        print("Art:", self._art,
              "\n    kjonn:", self._kjonn,
              "\n    vekt", self._vekt)
```

dyr.py

```python
from dyr import Dyr

hund = Dyr("hund", "hann", 10.2)
hund.skriv_info()

katt = Dyr("katt", "hunn", 5.7)
katt.skriv_info()

sau = Dyr("sau", "hann", 80.6)
sau.skriv_info()
```

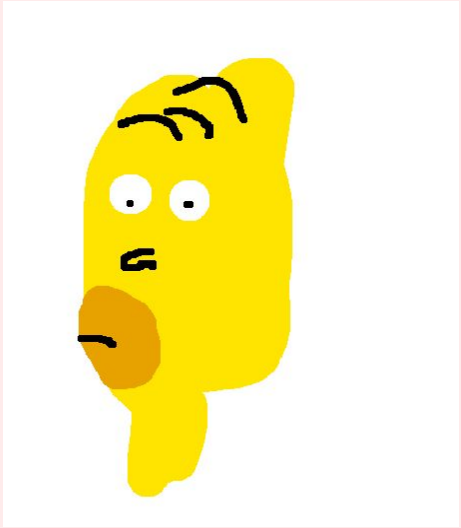
test_dyr.py

KONTAKT



...

sirisoll@uio.no
@sirisoll på Mattermost



UKENS MESTERVERK: “homer”

- ***doh***