in1000 uke 7 dypdykk

viktig forståelse fra forrige ukes pensum

læremål

0

lister / ordbøker

1

klasser: hvordan

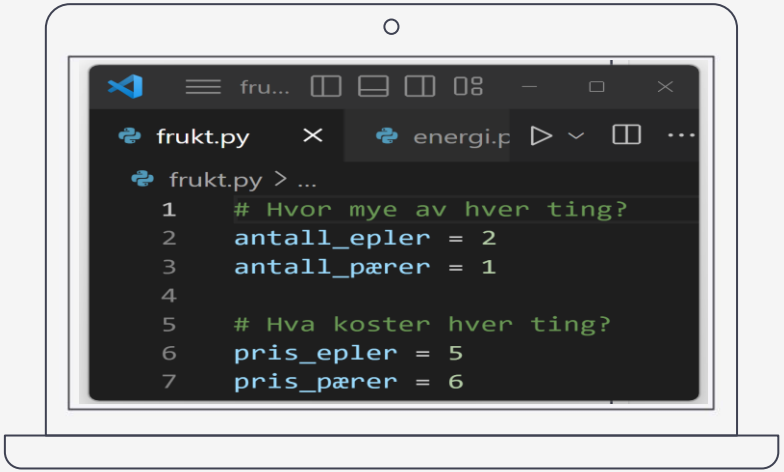
2

klasser: hvorfor

0

repetisjon

kort om lister og ordbøker



live-koding

lister og ordbøker i løkker

ordbokliste.py
(legges ut i timeplanen på emnesiden etter forelesningen)

1

hvordan

mer om klasser og objekter

hvorfor har konstruktøren ingen **return**?

```
def __init__(self, navn):
    self._navn = navn
    self._poeng = 0
```

```
student2 = Student("Tone")
```

det som *egentlig* skjer her (skjult for oss):

```
student2 = Student("Tone")
```

1) når vi kaller **Student(...)** lages **først** et nytt Student-objekt som returneres til variabelen student2

2) **etterpå** kjøres **__init__**, og nå har parameteren **self** det nylagde objektet som verdi

```
def __init__(self, navn):
    self._navn = navn
    self._poeng = 0
```

3) her trengs ingen **return**, for objektet ble returnert allerede i punkt 1

derfor gir dette feilmelding:

```
def __init__(self, navn):
    self._navn = navn
    self._poeng = 0
    return self # virker logisk, men gir feilmelding!
```

```
    student1 = Student("Endre")
TypeError: __init__() should return None, not 'Student'
```

2

hvorfor

klasser og objekter
innkapsling og grensesnitt

hva er poenget med å skjule detaljer i klasser/funksjoner?

- **oversikt**: programmet drukner ikke i detaljer

- **gjenbruk**: lettere å bruke i andre programmer senere

innkapsling:
å skjule detaljer i klasser og funksjoner (for oversikt og gjenbruk)

uten_innkapsling.py: er dette programmet lett å forstå?

```python
from random import randint

# Dette er hovedprogrammet (med alle detaljene)
barbarian = {
    "name": "Karsk",
    "attack_die": 20,
    "attack_bonus": 5,
    "damage_die": 12,
    "damage_bonus": 3
}

dragon = {
    "name": "Trogdor the Burninator",
    "defense": 15
}

for i in range(3): # repeter 3 ganger
    attack_roll = randint(1, barbarian["attack_die"])
    attack_total = attack_roll + barbarian["attack_bonus"]

    if attack_total >= dragon["defense"]:
        damage_roll = randint(1, barbarian["damage_die"])
        damage_total = damage_roll + barbarian["damage_bonus"]
        print(barbarian["name"], "hits", dragon["name"], "for", damage_total, "damage")
    else:
        print(barbarian["name"], "misses", dragon["name"])
```

med_innkapsling.py: er dette programmet lett å forstå?

```python
from random import randint

# Her ligger detaljene (skjult i klassene)
class Barbarian: ...

class Dragon: ...

# Dette er hovedprogrammet (lite og oversiktelig)
barbarian = Barbarian("Karsk")
dragon = Dragon("Trogdor the Burninator")

for i in range(3): # repeter 3 ganger
    barbarian.attack(dragon)
```

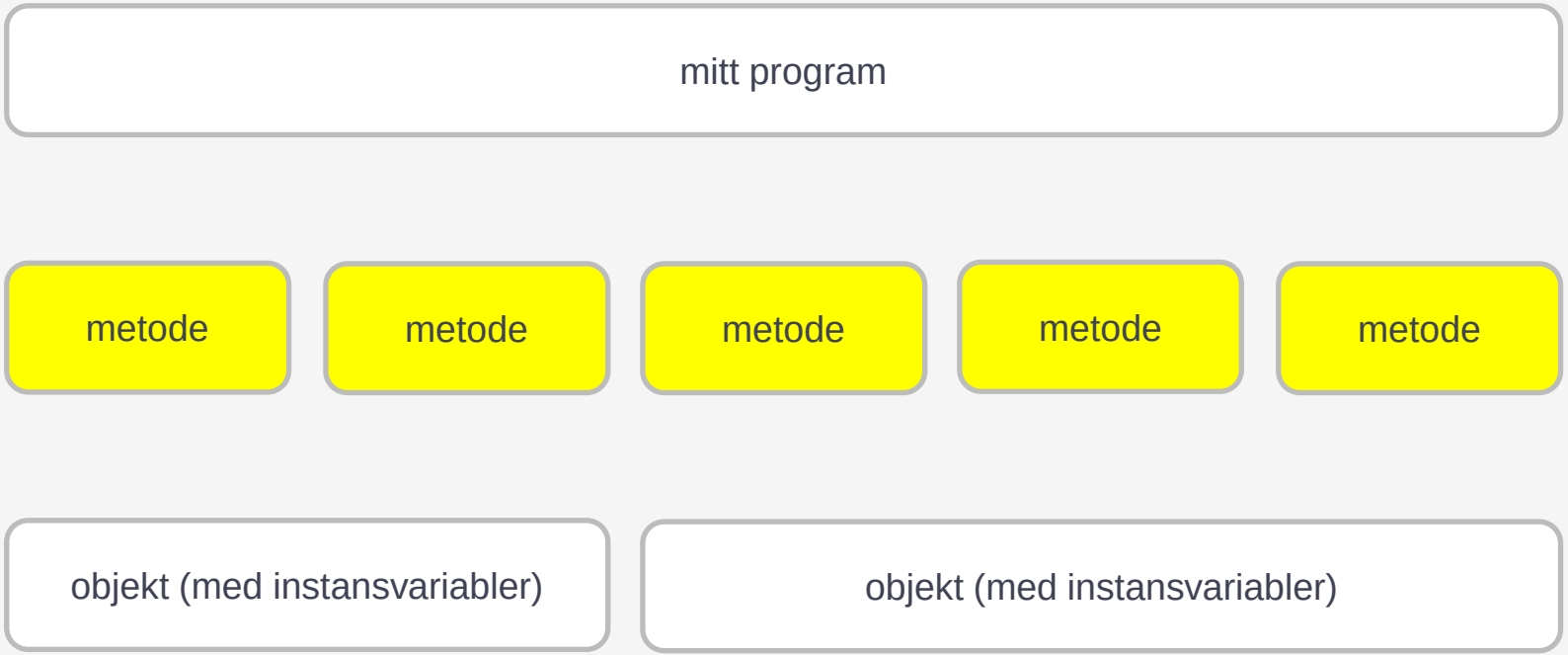
detaljer? de kan vi se på etter behov

```
# Her ligger detaljene (skjult i klassene)
class Barbarian:
    def __init__(self, name):
        self._name = name
        self._attack_die = 20
        self._attack_bonus = 5
        self._damage_die = 12
        self._damage_bonus = 3
```

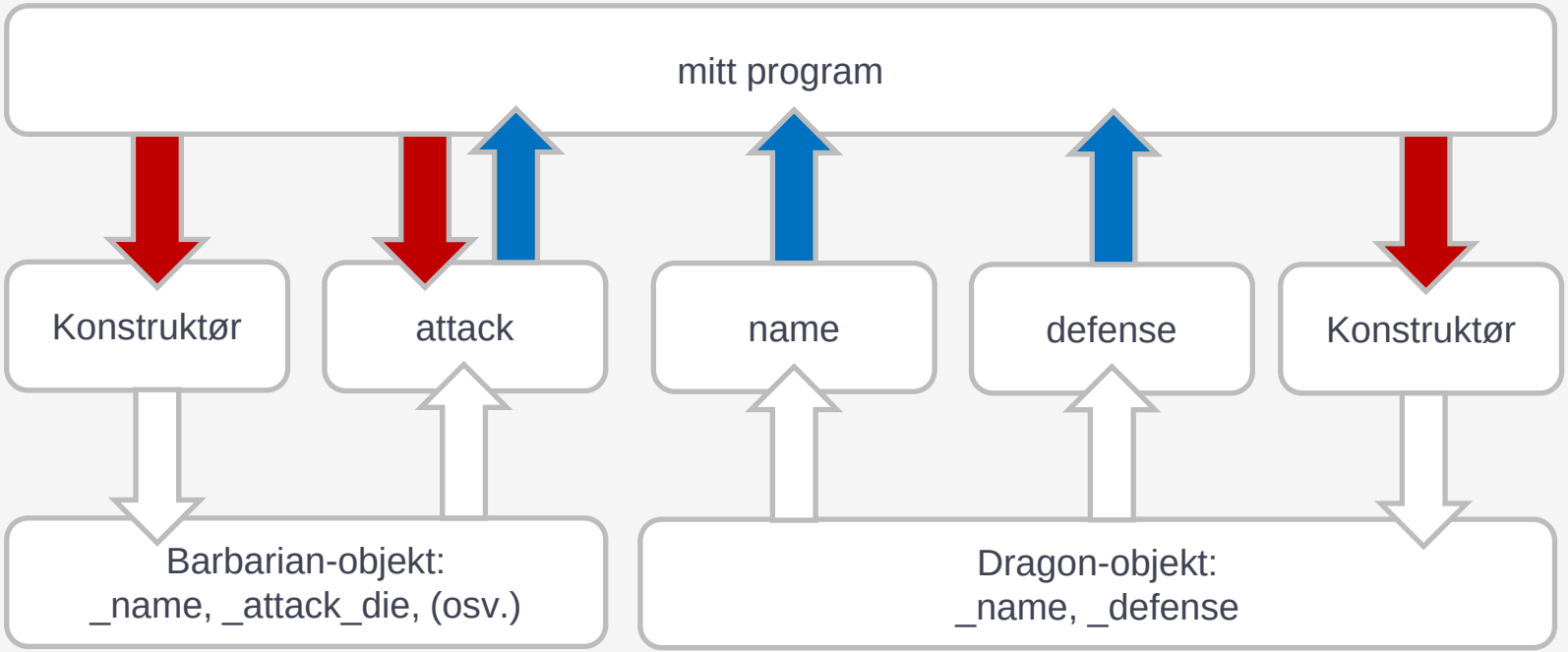
hvordan **kommuniserer** et program med noe som er kapslet inn?

- **inndata:** argumenter til metoder/funksjoner
- **utdata:** ta imot returverdier

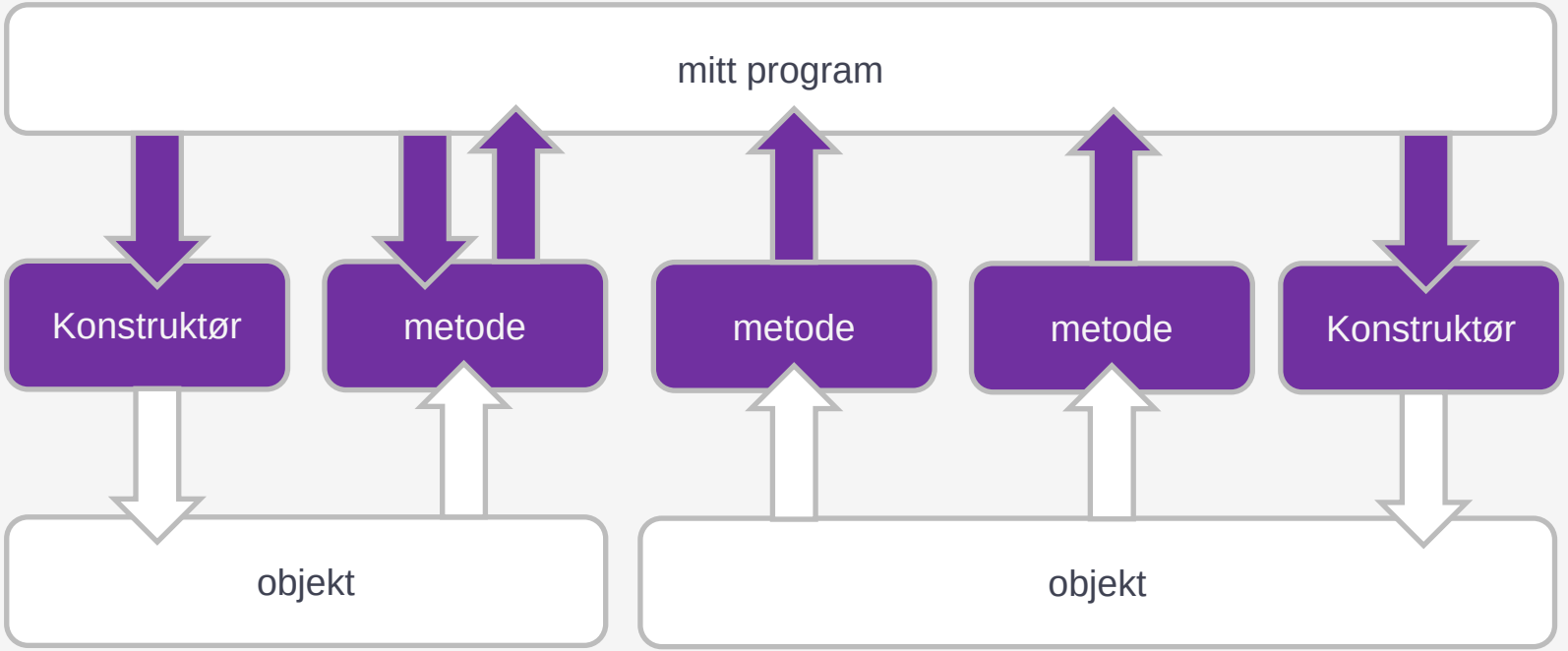
metoder sørger for kommunikasjonen mellom program og objekter



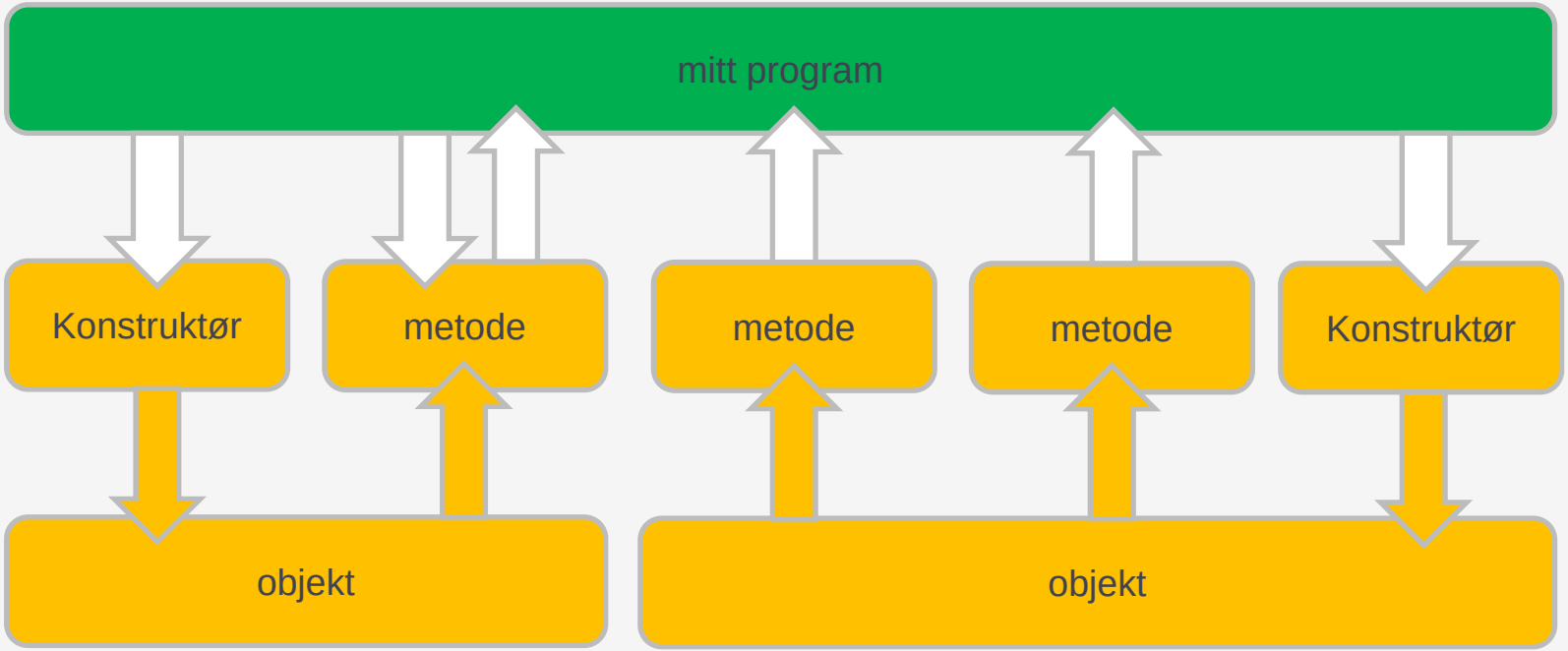
eksempel: argumenter og returverdier



grensesnitt: hvordan kommunisere med det som er kapslet inn (hva skal inn? hva kommer ut?)



hovedprogrammet tar seg av helheten
klassene tar seg av detaljene



eksempel på grensesnitt (tabeller)

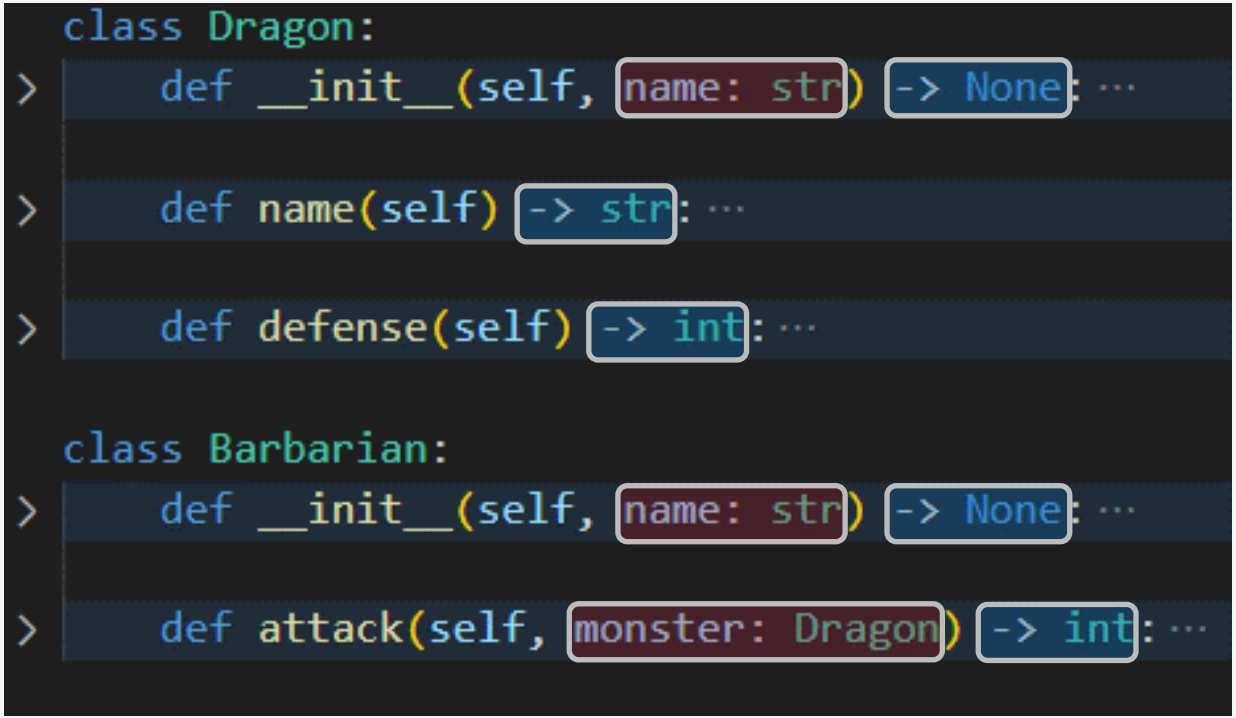
Dragon

metode	argumenter	returverdier
(Konstruktør)	name (str)	-
name	-	str
defense	-	int

Barbarian

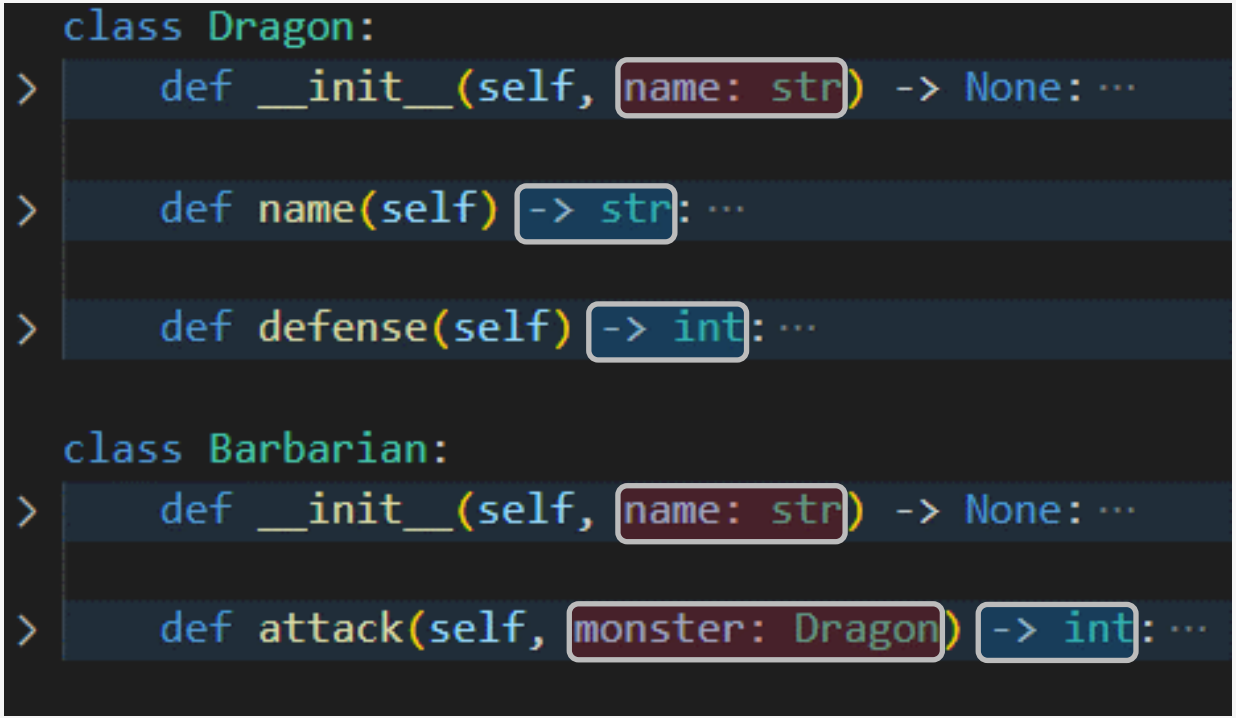
metode	argumenter	returverdier
(Konstruktør)	name (str)	-
attack	monster (Dragon)	int

eksempel på grensesnitt (frivillige **hint** til hva argumenter og returverdier vil være)



inndata til objekter og funksjoner

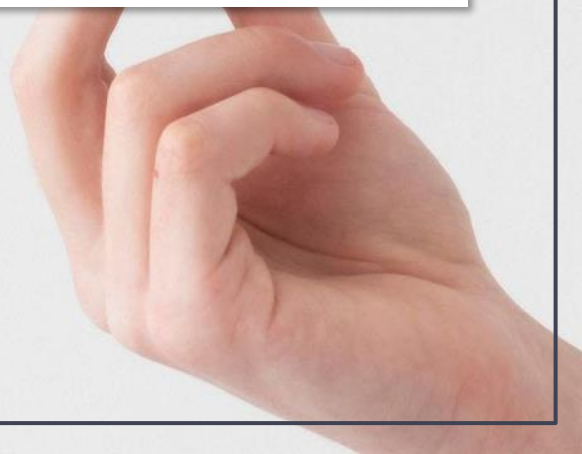
utdata tilbake fra objekter og funksjoner



mentimeter:

menti.com

kode: 73 67 48 2



in1000
uke 8
nytt lærestoff

oversikt og forståelse i kontekst

læremål

0

flere klasser

1

referanser

0

flere klasser

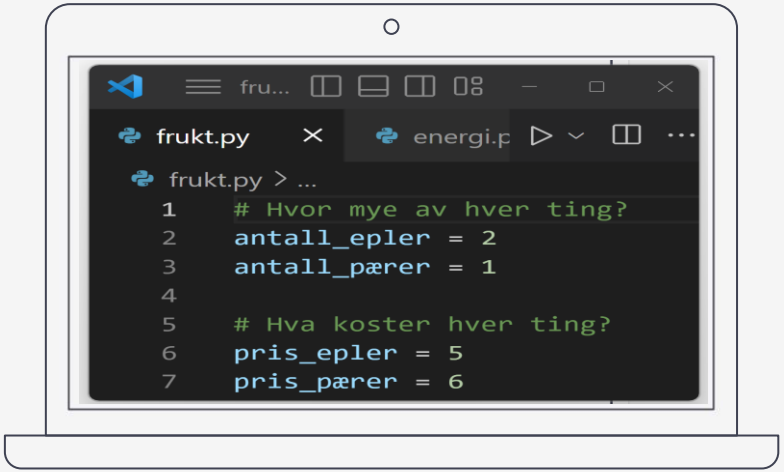
i ett og samme program

klassene kan defineres etter hverandre i samme fil:

```
# Her ligger detaljene skjult i klassene
class Dragon: ...

class Barbarian: ...

# Dette er hovedprogrammet (lite og oversiktelig)
barbarian = Barbarian("Karsk")
dragon = Dragon("Trogdor the Burninator")
```



live-koding

hvordan lage en klasse og objekter

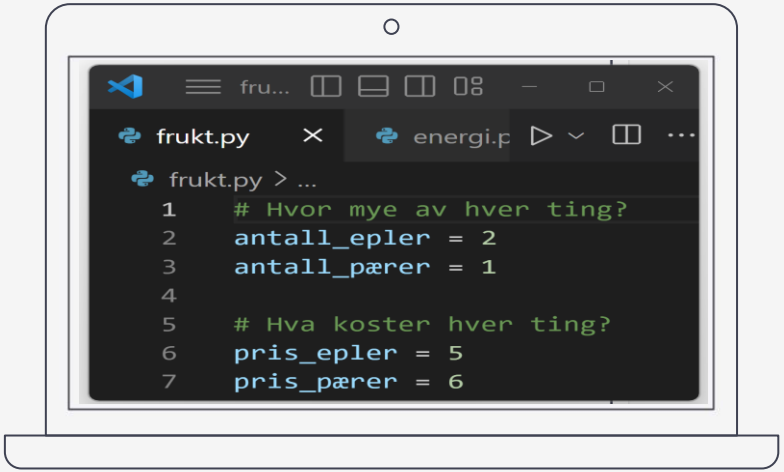
flereklasser.py
(legges ut i timeplanen på emnesiden etter forelesningen)

husk å se på koden i [python tutor]

1

referanser

objekter som refererer til andre objekter



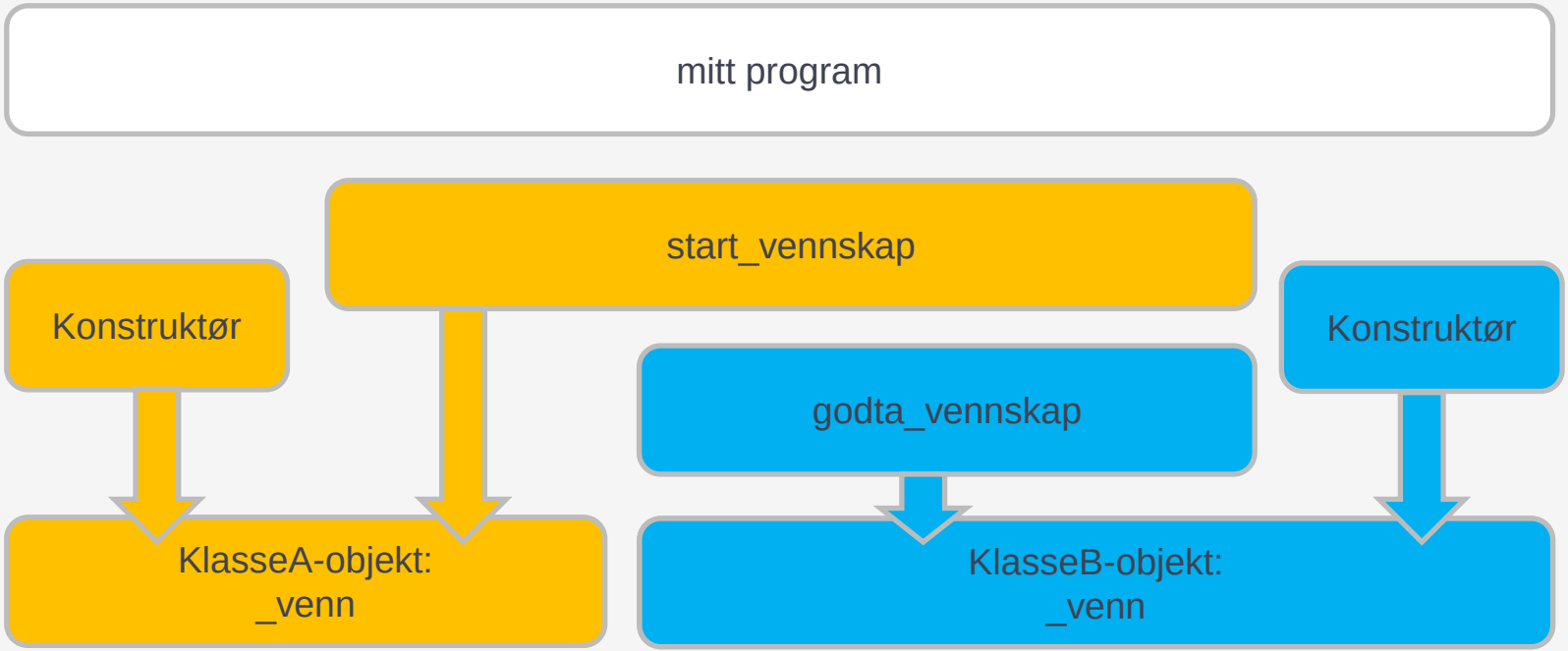
live-koding

hvordan lage en klasse og objekter

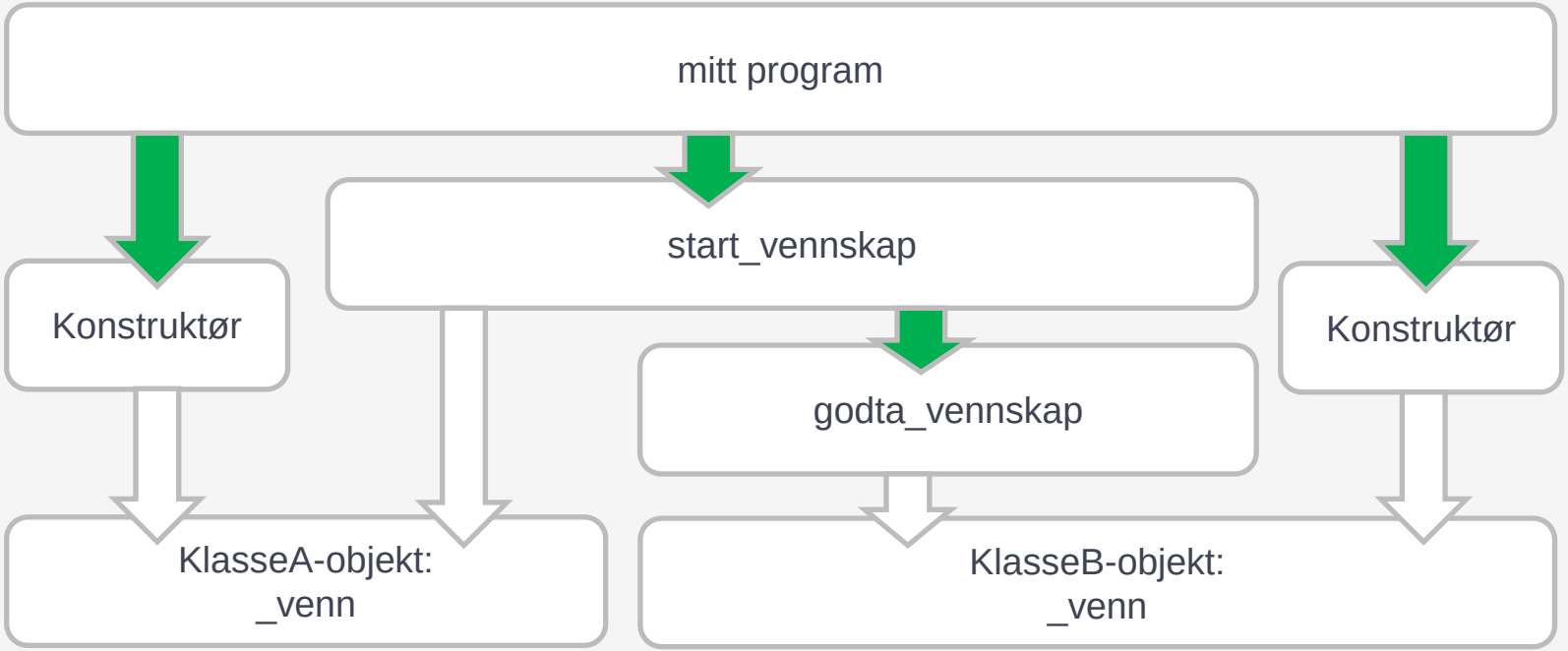
referanser.py
(legges ut i timeplanen på emnesiden etter forelesningen)

husk å se på koden i [python tutor]

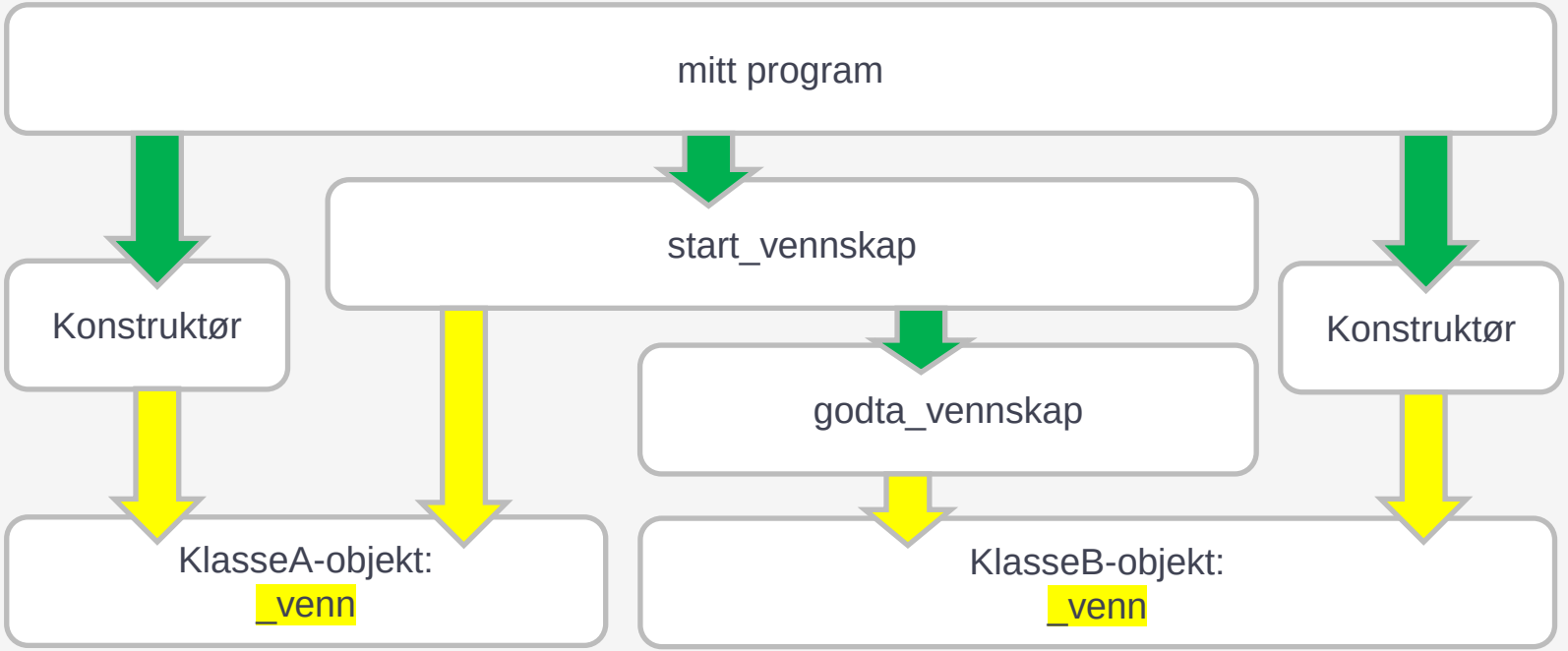
oversikt: hva tilhører klasseA og klasseB?



argumenter over flere nivå:
en metode kaller en annen metode



vi respekterer klassenes private variabler og endrer dem **kun** via grensesnittet (metoder)



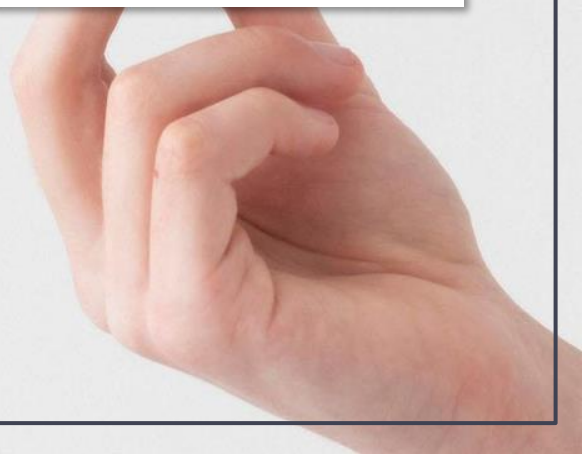
det som er **privat** er bare viktig for klassen selv

grensesnittet er derimot viktig for å **bruke** objekter fra denne klassen

mentimeter:

menti.com

kode: 6203 5601




slidesgo