Seminartime uke 9 – Gr. 10

IN1000 – HØST 2021

Litt praktisk

Noen lab-grupper er slått sammen

→ Gruppe 10 har sammen med gruppe 17 på torsdager

→ Samme tid og sted: Java, 1215-1400

Seminar-timer er slik som tidligere (tirsdager)

→ Gjerne send inn ønsker på opplegget

Hvor langt har vi kommet?

	Legges ut ca.	**Frist**
7. Innlevering OBLIGATORISK	13. oktober	26. oktober kl 23.59
8. Innlevering OBLIGATORISK	27.oktober	9. november kl 23.59

Eksamen: 3. desember

Uke	Tema
8	Mer objektorientert programmering - (Repetisjon objekter og klasser) - Referanser og objekter - Uforanderlige typer (immutable data types)
9	Mer komplekse strukturer - Objekter av flere klasser - Magiske metoder - Referanser mellom objekter
10	Mer komplekse strukturer - Egne beholdere for objekter - Mer komplekse datastrukturer med objekter
11	Problemløsing med klasser - Større eksempler - Løse problemer ved bruk av flere klasser
12	Problemløsing + prøveeksamen?
13+	Repetisjon og øve til eksamen ?

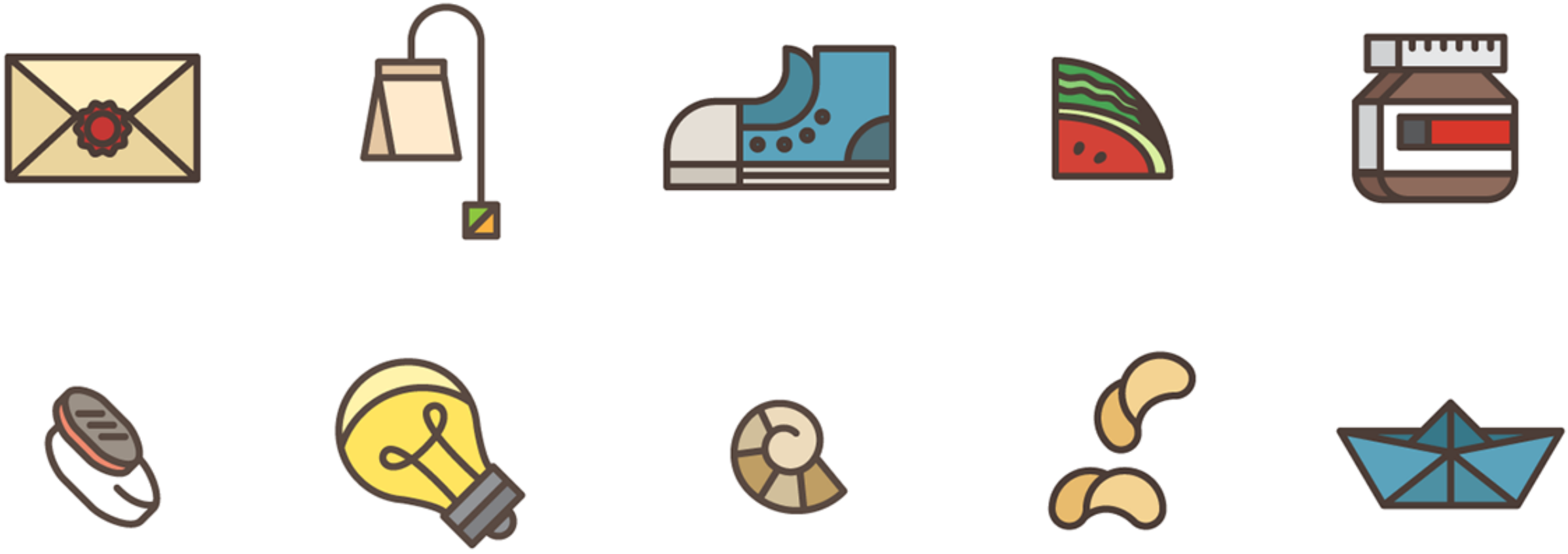
Plan for i dag

Del 1

- Menti (istedenfor Kahoot 😒)
- Litt repetisjon og diskusjon: klasser og OOP
- Magiske metoder
- Oppgave: Ekstraher og lag egne klasser (problemløsing) + datastrukturtegning

Del 2 (hvis tid)

- Egen jobbing/oppgaver



Repetisjon - klasser

Rep: klasser

- Hvorfor har vi en kontruktør __init__ ?
  → Noen verdier/egenskaper vi ønsker at ALLE objekter i klassen skal ha når de blir laget (typ «initial values»)
- Når skal vi ha/ikke ha parametre i konstruktøren?
  → Hvis instansvariabelen skal ha en fast verdi uansett, så trenge ikke det å bli sendt inn som parameter i konstruktøren. (Ønsker altså å sende inn som parameter det som varierer, eller det vi ikke vet fra før)
- Hvordan ønsker vi å bruke instansvariablene?
  → Brukes som kjennetegn/flagging? Brukes til å kunne endres? Til å lagre andre data?
- Hva ved objektet ønsker vi å åpne for endring?
  → Hvilke instansverdier skal vi gi «omverdenen» tilgang til å endre? F.eks kan «sult» endres på (forrige oblig)
- Når ønsker vi å returnere noe, vs å printe noe fra objektet?
  → Vil vi at omverdenen skal kunne bruke noe videre (retur)? Vil vi kun skrive ut en spesifikk ting på skjermen(print)?
- Hvordan ønsker vi at objektet skal være representert «utenfra»?
  → Hva vil vi skal returnere når vi kaller på objektet selv, f.eks i en print? (__str__)
- Hvordan ønsker vi å sammenlikne objektene?
  → Hva skal de sammenliknes på? Lengde? Verdi? Hvilken verdi? Hvordan definerer man likhet? (__eq__)

Magiske metoder

Magiske metoder

- Metoder som *ikke* blir kalt på slik vi er vandt til: **objekt.function()** eller **funkction()**
- Kallene skjer «internt» i den gjeldende klassen når vi skriver koden på en «spesiell» måte
- Mange innebygde klasser (int, string, bool….) i python har disse definert fra før
- I metodedeklarasjonen bruker man dobbel underscore på begge sider av navnet «__funjksjonsnavn__»
- Det betyr at __init__ også er en magisk metode!

Magiske metoder, forts.

La oss se på noen eksempler!

Klasse: int

Metode	Operator	Kall
object.__add__(self, **other**)	+	**2** + **2**
object.__sub__(self, **other**)	-	**45** – **34**
object.__mul__(self, **other**)	*	**11** * **10**
object.__mod__(self, **other**)	%	**4** % **2**
object.__eq__(self, **other**)	==	**23** == **23**
object.__iadd__(self, **other**)	+=	**100** += **1**

Dere skal implementere 2 stk: __eq__ og __str__

__eq__

- Forfatteren av klassene (dere) bestemmer hvordan objekter i klassen skal sammenliknes
- Vi har allerde skrivemåten for dette: __eq__ for metodekropp, og == for selve kallet
- Vi skal da lage *implementasjonen* til metoden

Eksempler på bruk (person):

- 2 person-objekter er like hvis de har like navn
- 2 person-objekter er like hvis de er like gamle
- 2 person-objekter er like hvis de har lik navn og alder
- 2 person-objekter er like hvis de har samme kjønn

Syntaks:

```
class MyList:

    def __init__(self):
        self._list = []


    def __eq__(self, other_list):
        if (len(self._list) == len(other_list)):
            return True
        else:
            return False
```

__str__

- Skiller seg fra metoder som __eq__ , tar ikke inn et annet objekt
- Denne metoden bestemmer hvordan et objekt «ser ut» når det printes
- Tenke på: hva er naturlig å skrive ut, hvilke instansvariabler kan man bruke?
- Returnerer alltid en streng

Før

```
from Hund import Hund

ny_hund = Hund("Fido", 3, 23, "Golden")
print(ny_hund)
```

```
C:\Users\vinhp\Desktop\Alt\Informatikk\Første semest
<Hund.Hund object at 0x000001CDBA7EA7F0>
```

Etter

```
from Hund import Hund

ny_hund = Hund("Fido", 3, 23, "Golden")
print(ny_hund)
```

```
C:\Users\vinhp\Desktop\Alt\Informatikk\Først
Fido
```

__str__

```python
class Hund:
    def __init__(self, navn, alder, vekt, rase):
        self._navn = navn
        self._alder = alder
        self._vekt = vekt
        self._rase = rase
        self._favorittmat = []
```

Syntaks:

```
def __str__(self):
    return <hvilken som helst streng>
```

Eksempel 1

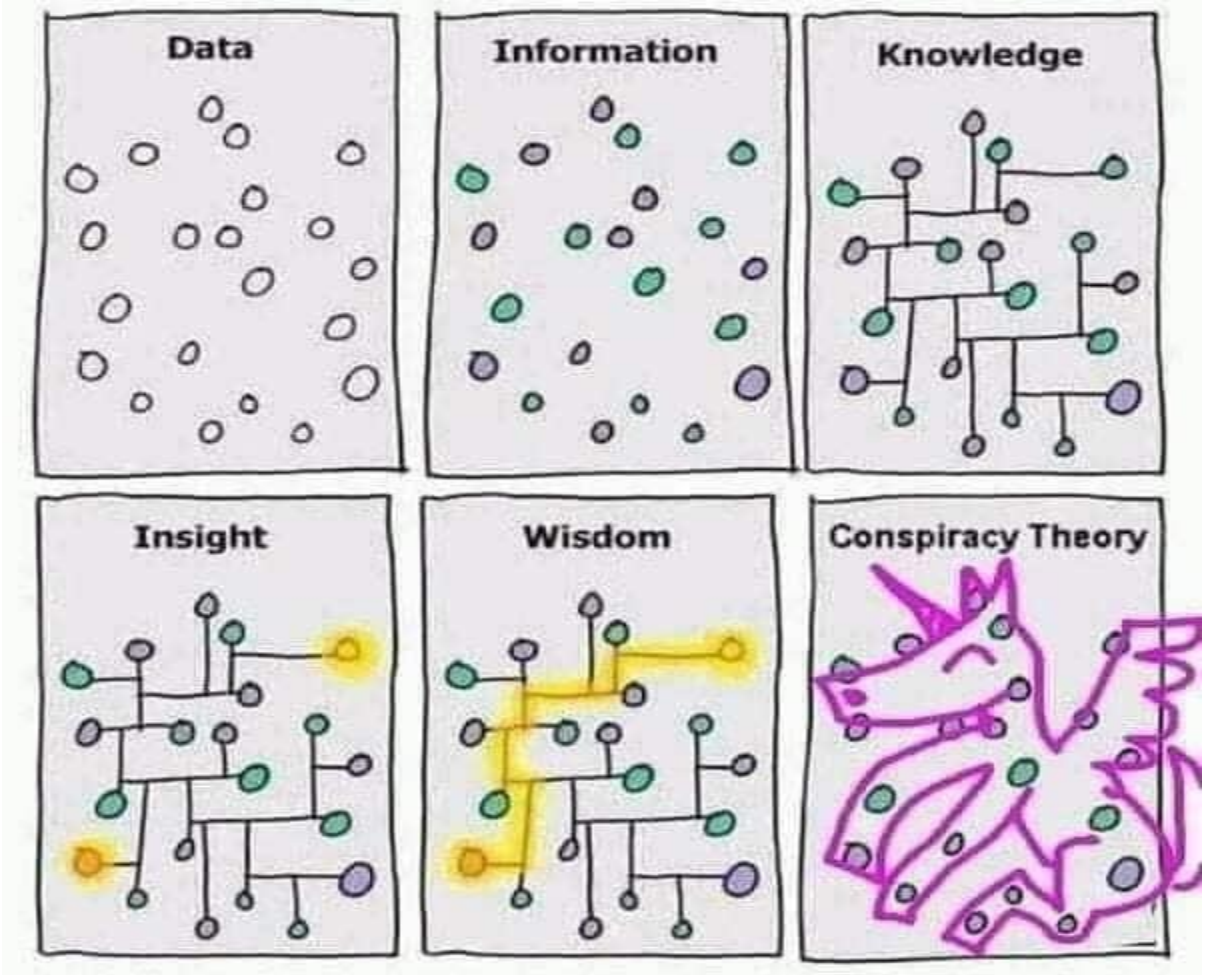
```python
def __str__(self):
    return self._navn
```

Eksempel 2

```python
def __str__(self):
    return "Navnet til hunden er " + self._navn
```

Eksempel 3

```python
def __str__(self):
    return "Dette er et hunde-objekt"
```



Oppgaveløsing med datastrukturtegning

Strukturtegninger i IN1000

Variabler med de «vanlige» typene (string, int, boolean….)

<variabelnavn>

<verdi>

alder

18

Objekter av (egne) mutable klasser

:<klassenavn>

<instansvariabel> = <verdi>

: Student

_antDeltatt = 0

Variabler som refererer til objekter

<variabelnavn>

:<klassenavn>

<instansvariabel> = <verdi>

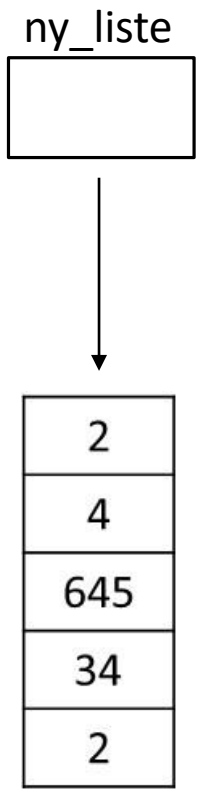
nyDeltaker

: Student

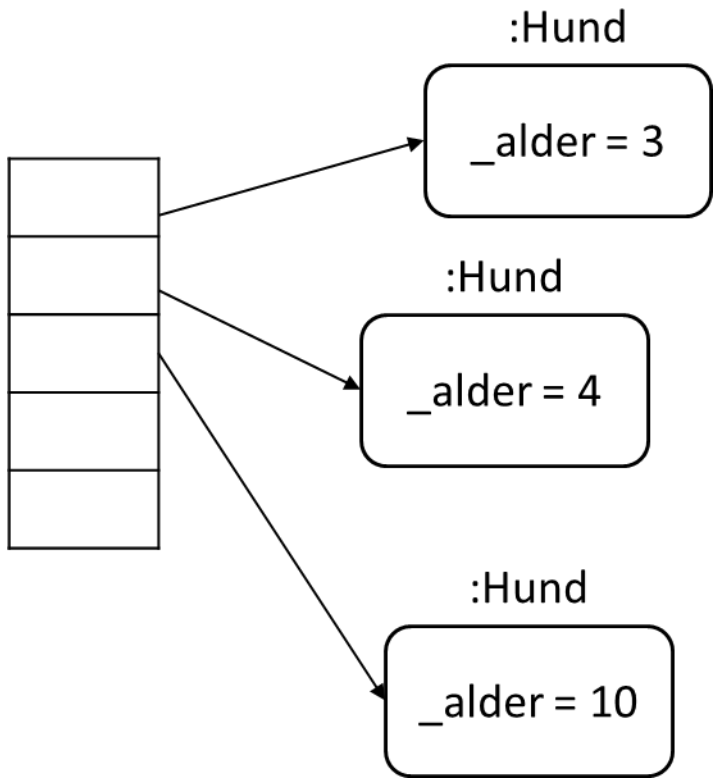
_antDeltatt = 0

Strukturtegninger i IN1000 forts.

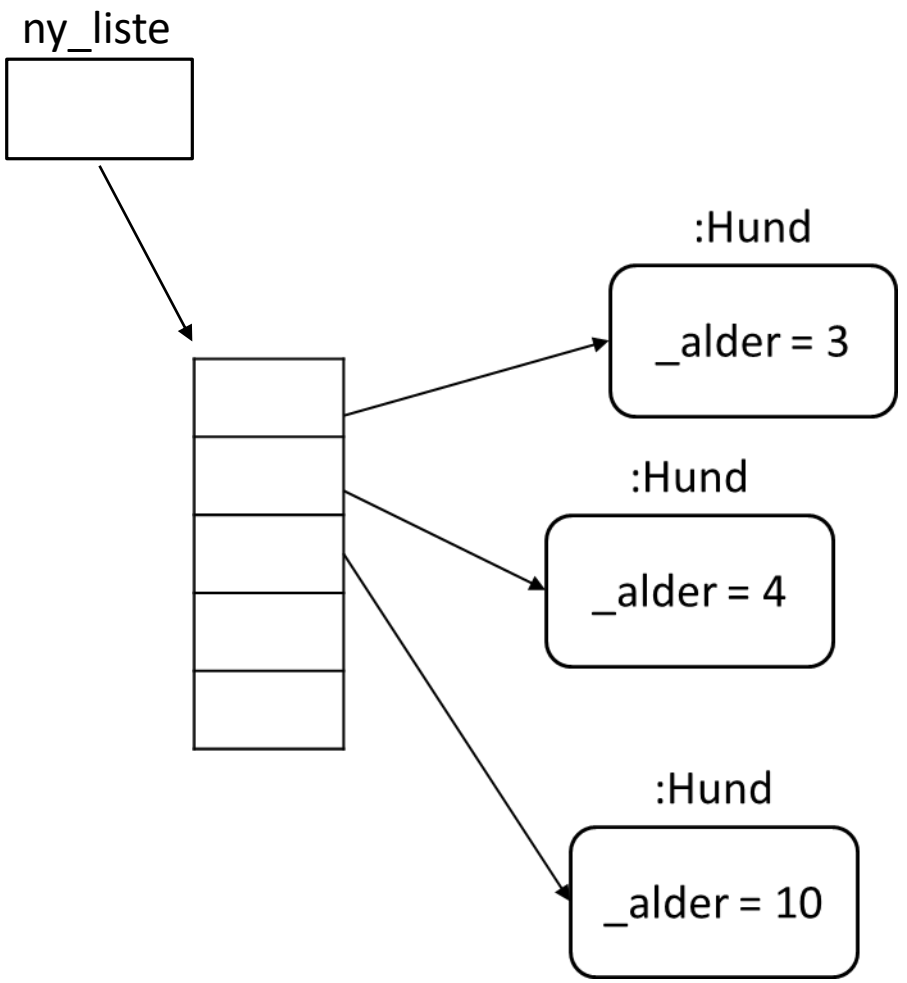
Lister med «vanlige» datatyper (string, int, boolean….)



Lister med objekter



Variabel som peker på lister med objekter



Oppgave

- Dere får en tekst av en «situasjon» / «case»

- Deres oppgave er å lage en representasjon av caset i python

- Dere skal (del 1):
  - Identifisere/finne aktuelle klasser og datatyper som inngår
  - *modellere* caset med en datastrukturtegning

- Så skal dere (del 2, mulig neste gang):
  - *Programmere* denne strukturen i python
  - Lese fra fil for å lage objekter, og teste at programmet fungerer

- Felles gjennomgang/løsningsforslag av del 1

Oppgavetekst

Galtvort Høyere skole for hekseri og trolldom er en trollmannskole som underviser i magi. For å skjule borgen er skolen er, har man vært nødt til å kaste de kraftigste skjuleformlene som finnes, noe som vetyr at skolen er non-kartografisk. Galtvorts elever blir delt inn i fire hus: Griffing, Ravnklo, Håsblås og Smygard. Husene ble navngitt etter de fire grunnleggerne av skolen. Det er ca 1000 elever ved skolen, og man antar at det er ca 40 elever per trinn (1.-7.) i hvert hus.

Griffing er kjent for å ha folk med: tappert hjerte, vågemot og edelmot. Husets farger er rød og gull, og er symbolisert med en løve.

Ravnklo er kjent for å ha folk med: Kløkt, lærdom og trang til kunnskap. Husets farger er blå og bronse (i filmen blå og sølv), husets symbol er en ravn.

Håsblås er kjent for å ha folk med: Tålmodighet, lojalitet og hardt arbeid. Husets farger er sort og gul, og huset er symbolisert med en grevling.

Smygard er kjent for å være det huset hvor folk som er slue og vrir alt til sitt eget beste befinner seg eller har vært i. Husets symbol er en slange og fargene er grønn og sølv.

Oppgavetekst – utvidelse (vanskelig)

For hvert trinn er det en ferdig definert liste med pensum-bøker. Alle elever i det gjeldende trinnet skal ha de riktige pensumbøkene. Dersom det er en endring i pensumet, skal alle elevene få det med seg.
Når en elev bytter klassetrinn, skal pensumlsiten til eleven også skifte til pensum for riktig trinn.

Utvid det forrige programmet med en ny klasse som representerer pensumet for hele Galtvort. Husk også at det aktuelle pensumet skal være knyttet til hver elev.