

IN1000 Forelesning 7

Hvordan bruke objekter i Python?

Anbefalte innstillinger til VS Code

https://www.uio.no/studier/emner/matnat/ifi/IN1000/v26/Materiale-fra-forelesninger-mm/settings.json

Livekoding: Roller til variabler

Livekodingene er viktig pensum og relevante for eksamen

Vi skriver koden sammen her i forelesningen

Rekk opp en hånd hvis det går for fort

Hvis du ser forelesningen i opptak, sett den på pause når du trenger og skriv koden sammen med oss som var her

Hvis det er noe du ikke får til å virke, ikke stopp opp for å fikse feilen – se på det hjemme eller i gruppetimene

Hva skal vi med funksjoner?

- Hensikten er å gjøre programmer mer oversiktelige og lettere å skrive og lese for mennesker
- Funksjoner et er verktøy til å redusere kompleksitet
- Vi kan *bruke* funksjoner uten å tenke på *hvordan* de virker innvendig
- *print*-funksjonen i Python er egentlig 100 linjer skrevet i C (et annet programmeringsspråk), men det har ikke vi trengt å tenke på – det viktigste for oss er å bruke den riktig
- **Hva** funksjonen gjør og **hvordan den brukes riktig** er viktig når vi bruker en funksjon
- **Hvordan** funksjonen gjør det den gjør er bare viktig når vi lager / endrer funksjonen

Et nytt læremål

Hva lærer du?

Etter å ha tatt IN1000:

- forstår du prinsippene for objektorientert programmering og kan benytte disse til å skrive enklere objektorienterte programmer

Hva skal vi med objekter?

- Hensikten er å gjøre programmer mer oversiktelige og lettere å skrive og lese for mennesker (akkurat som med funksjoner)
- Også objekter et er verktøy til å redusere kompleksitet
- Vi kan *bruke* objekter uten å tenke på *hvordan* de virker innvendig
- Litt som at vi kan bruke en bil uten at vi trenger å kunne bygge eller reparere en
- **Hva** objektet gjør og **hvordan det brukes riktig** er viktig når vi bruker et objekt
- **Hvordan** objektet gjør det det gjør er bare viktig når vi lager / endrer oppskriften på hvordan denne typen objekt skal lages

Hva er viktigst? Detaljer eller helheten?

- Man kan definitivt få førerkort uten å være bilmekaniker og kjenne alle detaljene i bilen
- Det er nok også mulig å være bilmekaniker uten å ha førerkort, men er det lurt?
- Det er **vanskeligere** å bygge og reparere noe som man ikke aner hvordan skal brukes – fordi man virkelig trenger å kunne se ting fra brukeren sitt perspektiv
- Vi starter med det viktigste: **grensesnittet** mellom oss og bilen (pedaler, ratt, girspak)
- For å forstå en bil er det viktigste å forstå
  - Hvordan vi bruker den riktig (**input**)
  - Hvordan bilen da responderer på det vi gjør (**output**)
- Først da kan vi gå til de små detaljene (hva skjer inni motoren, hvordan virker giret...)
- **Vi kan forstå det viktigste (på høyt nivå) uten å forstå hver minste detalj (lavt nivå)**

Fra biler til objekter generelt

- Man kan definitivt bruke objekter uten å ha laget oppskriften og kjenne alle detaljene
- Det er nok også mulig å være lage oppskriften uten å forstå hvordan objektet kan brukes fornuftig, men det gjør eksamen i IN1000 unødvendig vanskelig for mange
- Det er **vanskeligere** å bygge og reparere noe som man ikke aner hvordan skal brukes – fordi man virkelig trenger å kunne se ting i et større perspektiv enn detaljene
- Vi starter med det viktigste: **grensesnittet** mellom oss og objektet (instansvariabler, metoder)
- For å forstå objekter er det viktigste å forstå
  - Hvordan vi bruker dem riktig (**input**)
  - Hvordan objektet da responderer på det vi gjør (**output**)
- Først da kan vi gå til de små detaljene (hvordan virker den metoden, hvordan er objektet laget, hvordan ser det i minnet på datamaskinen)
- **Vi kan forstå det viktigste (på høyt nivå) før vi forstår hver minste detalj (lavt nivå)**

Livekoding: Hva *er* et objekt egentlig?



Du trenger filen eksempel.py fra emnesiden
(i mappen forelesning_07)

Objekter inneholder data

- Et objekt kan ha mange instansvariabler med data
- På denne måten kan vi samle data som hører sammen i ett objekt
- Da kan vi jobbe med dette objektet i stedet for å jobbe med datavariablene hver for seg
- I stedet for
  - en variabel med symbolet til kortet (str), for eksempel “kløver”
  - en variabel med tallverdien til kortet (int), for eksempel 4
- Så har vi ett *Spillkort*-objekt som inneholder alt dette
- Lettere å se helheten, ikke bare detaljene!

Objekter er mer enn bare passive data – de kan også gjøre ting

- I tillegg til data kan objekter gjøre ting selv
- Ikke fordi de er intelligente, men fordi vi (eller noen) har laget en oppskrift på hvordan de skal gjøre det
- For eksempel vet en liste hvordan den skal sortere seg selv (så lenge innholdet i listen er av en type som kan sorteres)
- Å gi objektet til en **funksjon** er som å behandle objektet som en baby som ikke klarer å gjøre noe selv
  - for eksempel: *sorted(listeobjekt)* *# hei, funksjon, sortér dette & gi meg resultatet!*
  - Men da trenger vi jo en funksjon i tillegg til selve objektet…
- Å selv gi objektet ansvar for å kunne gjøre ting på egen hånd er som å behandle det som et større barn, som kan gjøre ting på egen hånd
  - for eksempel: *listeobjekt.sort()* *# hei, listeobjekt, sortér deg selv!*

Hvordan ser vi forskjell på funksjoner og metoder?

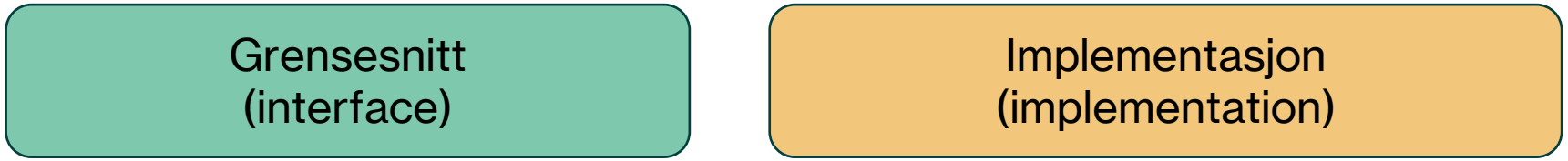
- *funksjon(objekt)*
- *objekt.metode()*
- Ikke bli forvirret av at metoder også kan ta inn ekstra argumenter!
  - *funksjon(objekt, annet_objekt)*
  - *objekt.metode(annet_objekt)* *# “objekt” er her det aktive objektet som gjør noe*
    # “annet_objekt” er passivt argument til metoden
- Eksempel:
  - *tall = HjemmelagetHeltall(10)*
    tall2 = HjemmelagetHeltall(20)
    tall3 = tall.pluss(tall2) *# samme som: tall3 = HjemmelagetHeltall(30)*
- (Legg merke til at **navnene** på typen *HjemmelagetHeltall* og metoden *pluss* forteller oss **hva** som skjer selv om vi ikke vet **hvordan** disse er laget! Det er lurt!)

Har det så mye å si om vi bruker funksjoner eller metoder?

- Det er mindre komplekst å importere en type objekt som kan gjøre noe selv, enn både en type objekt og en funksjon som skal gjøre noe med dette objektet
- Vi kan si at med metoder er ting bedre pakket inn – vi trenger bare importere én ting
- Funksjoner skal kanskje fungere for mange ulike typer objekter samtidig!
- Så da må funksjonen kanskje tenke på at dette må gjøres forskjellig måte for hver type objekt
- Med metoder vet vi akkurat hva slags type objekt dette er, så vi trenger ikke ta hensyn til hva andre typer objekter trenger å gjøre
  - (unntak: noe som heter **arv** – men dette er ikke pensum i IN1000 ☺)

Objektorientering er en måte å *tenke* på

- For datamaskinen har det lite å si om vi bruker objekter eller ikke – maskinkoden er ikke så forskjellig
- Men for oss mennesker kan denne måten å tenke på (dataobjekter som kan gjøre ting selv) gjøre programmene enklere å skrive, lese og forstå
- Hjelper oss å få oversikt over store programmer
- Skiller mellom hva objektet gjør (sett utenfra) og hvordan det gjør det (sett innenfra)

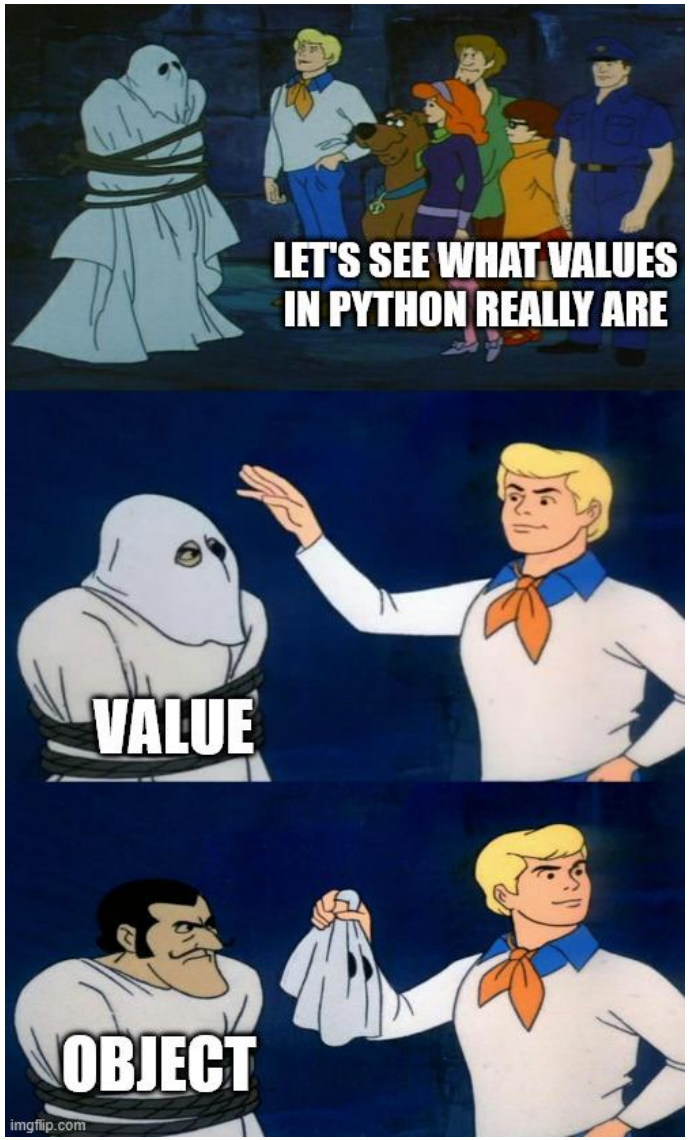


- Alt vi har gjort i IN1000 til nå har vært å lære å bruke grensesnittet til noen typer objekter riktig

Innkapsling

- Det viktige for å bruke klassen rett er i **grensesnittet**
  - metoder (med parametre, returverdier)
  - instansvariabler
  - dokumentasjon
- Detaljene som bare er viktig internt er **innkapslet** (“gjemt bort”) inne i objektet (**implementasjonen**), så vi kan bruke objektet uten å tenke på disse
  - men vi må inn her hvis vi skal endre på / reparere hvordan det virker

Ja, du har jobbet med objekter hele tiden:



<u>De mest grunnleggende (innebygde) objektene:</u>

- Heltall er objekter (av typen **int**)
- Desimaltall er objekter (av typen **float**)
- Tekststrenger er objekter (av typen **str**)
- Sannhetsverdier (*True* og *False*) er objekter (av typen **bool**)

<u>Samlinger som inneholder flere elementer:</u>

- Lister er objekter (av typen **list**)
- Ordbøker er objekter (av typen **dict**)

Alle variabler i Python peker på objekter

- Når objektene har instansvariabler, vil disse inneholde andre objekter, som igjen kan ha instansvariabler som peker på andre objekter osv. osv...
- Hvor stopper denne kjeden? Når kommer vi til “slutten”?
- De “primitive” typene **int, float, str,** og **bool** har ikke instansvariabler, så det stopper med dem – disse typene inneholder verdier som ligger direkte i minnet på datamaskinen
- Ellers oppfører de seg som andre objekter

Innebygde objekter er litt spesielle

- Det finnes **snarveier** i programmeringsspråker til å lage dem (fordi de brukes veldig ofte):
  - *svar = 42*
  - *prosent = 1.337*
  - *navn = “Horwath Slesk”*
  - *suksess = True*
  - *liste = [3, 2, 1]*
  - *ordbok = {“nøkkel1”: “verdi1”, “nøkkel2”: “verdi2”}*
- Men vi **kan** lage dem uten å bruke snarveiene (og dette virker for **alle typer**):
  - *svar = int() # får standardverdien 0*
  - *liste = list() # får standardverdien [], altså en tom liste*

Syntaks for objekter, oppsummert

- Hvordan finner vi typen til et objekt (når *x* er en variabel)?
  - *print(“x har typen”, type(x))*
- Hvordan lager vi et nytt objekt av typen (eller klassen) *HjemmelagetObjekt*?
  - *x = HjemmelagetObjekt()*
- Noen typer objekter må i tillegg ha **argumenter** for å lages, akkurat som noen funksjoner må ha det (det går an å tenke på *HjemmelagetHeltall* som en slags funksjon som returnerer et nytt objekt av typen *HjemmelagetHeltall*)
  - *y = HjemmelagetHeltall(100)*
- Objekter kan ha variabler (**instansvariabler**): *z = y.verdi*
- Objekter kan ha egne funksjoner (**metoder**) som lar objektet gjøre noe: *y.doble_verdi()*

Oblig 6 (publiseres denne uken)

- Dere skal bruke objekter som dere ikke har laget enda
  - (disse lager dere først i oblig 7)
- For å likevel kunne kjøre deres egen oblig 6-kode uten feilmelding har vi laget noen enkle og uferdige prototyper av objektene som skal brukes, som gjør minst mulig
- Det vil si at dere i mindre grad enn vanlig får testet at koden deres i oblig 6 virker ved å kjøre programmet (fordi objektene foreløpig ikke gjør det de egentlig skal)
- Dette vil føles uvant, men dere vil få testet koden som normalt i oblig 7 – poenget med oblig 6 er ikke at koden skal være feilfri, men at dere klarer å bruke objekter

Neste uke

- Hvordan lager vi egne hjemmelagde objekter?
- Hvordan kan vi bruke egne objekter til å gjøre programmer mer oversiktelige?
- Det vil si skille mellom **hva** objektet gjør og hvordan det **brukes**... (sett **utenfra**)
  - (dette er det vi har sett på i dag, som vi fortsatt skal jobbe videre med)
- ...og detaljene i **hvordan** det gjør disse tingene (sett **innenfra**)
  - (dette begynner vi å se på neste gang)