Uke 7, time 1

Onsdag 1. oktober

Introduksjon til objektorientert programmering

Bakgrunn

Klasser, objekter og metoder

- Klasser og klassedefinisjoner
- Objekters liv, konstruktører
- Metoder og tjenester
- Instanser og deres variabler

Grensesnitt og innkapsling

Design, implementasjon og testing

Paradigmer

Imperativ (instruksjoner – som endrer tilstand)

- Prosedyrisk
- Objektorientert

Deklarativ (ønsket utfall)

- Funksjonell
- Logisk
- Reaktiv

Beskrivende (hardware)

- Strukturell
- Oppførsel
- Dataflyt

Domene (domenespesifikk)

- Struktur og styling

Hvordan tenker jeg på og organiserer programmet mitt

OOP – Klasser, objekter og metoder

Klasse

- Beskrivelsen, plantegningen, tilhørigheten eller "DNA"et til objektene

Klassedefinisjon

- Tjenester gjennom offentlige metoder (grensesnittet)
- Implementasjonen av metodene
- Data og struktur lagret i objektet → instansvariabler

OOP – Klasser, objekter og metoder

Objektets livsløp

- Opprettes (`__new__`)
- **Konstruereres** (`__init__`)
- Bruk
- Referanseteller
- "Søppeltømming"
- Dekonstrueres (`__del__`)

Objekter er strukturer som vi lager etter behov basert på klassen

Objekter kalles også for **instanser**

OOP – Klasser, objekter og metoder

Metoder er veldig lik prosedyrer og funksjoner som vi har brukt til nå, men de defineres i klassen, og tilhører og brukes i objektene

Vi deler metoder som enten offentlig eller privat

De offentlige metodene bestemmer **grensesnittet** til klassen, og forteller hvilke tjenester som objektene kan gjøre eller "levere"

Private metoder tilhører implementasjonen og brukes bare internt i klassen/objektet, vi sier at disse er **innkapslet**

I Python så er det ikke et eksplisitt skille mellom disse, men den mest vanligste måten for å vise hva som er hva, er å la de private metodene begynne med **én understrek** i metodenavnet

OOP – Klasser, objekter og metoder

Instansvariabler er som vanlige variabler, men hvor vi lagrer verdier som tilhører objektet (altså den opprettede instansen)

Vi definerer hvilke vi vil ha inne i konstruktøren, men må gis verdi i hvert objekt/instans

På samme måte som metodene, så kan vi bruke de både offentlig og privat, og vi markerer det likt med én understrek

– men, instansvariabler bør alltid være private

OOP – Klasser, objekter og metoder

Konstruktøren er i Python en metode som brukes for å opprette objektet. Teknisk kan vi kalle det initialisering, og metoden heter alltid `__init__`

Den trenger alltid minst en parameter – self –, men kan også ha flere parametere som vi kan bruke ved opprettelsen av objektet. Men når vi lager nye objekter så blir ikke self en del av parametrene vi trenger å oppgi

Vi har allerede brukt klasser til å opprette objekter:

```
str(2) # oppretter et string objekt av tallet 2
list(range(5)) # liste-objekt av 0, 1, 2, 3, 4
```

OOP – Grensesnitt og innkapsling

Den viktigste grunnen til at vi bruker OOP er at vi ønsker modularisering og en struktur for programmet som passer en modell vi har definert.

Vi oppnår det ved å lage et offentlig **grensesnitt** som kan brukes av annen kode

Men for at modulariseringen skal fungere, så må vi holde grensesnittet så veldefinert og minimalt som mulig

– og ikke trenge å tenke på hva som skjer innenfor objektet

For annen kode, så skal klasser, objekter, og metodene fungere som "svarte bokser" som vi ikke trenger å tenke på hvordan fungerer

OOP – Grensesnitt og innkapsling

Dette kaller vi **innkapsling**, og betyr at vi ikke bruker private metoder, eller forandrer på instansvariabler

Med andre ord, aldri bruk direkte eller kjør variabler og metoder som begynner på _ i navnet utenfor objektet/klassen du er i

OOP – self

Alle metoder må ha `self` som første parameter

`self` viser tilbake til objektet

Vi må bruke `self` for å bruke instansvariablene i objektet



OOP – Eksempler klasser

```
class Bok:
    def __init__(self, tittel, forfatter, sider):
        self._tittel = tittel
        self._forfatter = forfatter
        self._antall_sider = sider
        self._er_lest = False

    def les_bok(self):
        # lese, lese, lese
        self._er_lest = True

    def har_jeg_lest(self):
        return self._er_lest

    def hent_info(self):
        return f"Tittel: {self._tittel}, forfatter: {self._forfatter}"
```

OOP – Eksempler klasser

```
class Bok:
    def __init__(self, tittel, forfatter, sider):
        self._tittel = tittel
        self._forfatter = forfatter
        self._antall_sider = sider
        self._er_lest = False

    def les_bok(self):
        # lese, lese, lese
        self._er_lest = True

    def har_jeg_lest(self):
        return self._er_lest

    def hent_info(self):
        return f"Tittel: {self._tittel}, forfatter: {self._forfatter}"
```

Konstruktør

OOP – Eksempler klasser

```
class Bok:
    def __init__(self, tittel, forfatter, sider):
        self._tittel = tittel
        self._forfatter = forfatter
        self._antall_sider = sider
        self._er_lest = False

    def les_bok(self):
        # lese, lese, lese
        self._er_lest = True

    def har_jeg_lest(self):
        return self._er_lest

    def hent_info(self):
        return f"Tittel: {self._tittel}, forfatter: {self._forfatter}"
```

Metoder

OOP – Eksempler klasser

```
class Bok:
    def __init__(self, tittel, forfatter, sider):
        self._tittel = tittel
        self._forfatter = forfatter
        self._antall_sider = sider
        self._er_lest = False

    def les_bok(self):
        # lese, lese, lese
        self._er_lest = True

    def har_jeg_lest(self):
        return self._er_lest

    def hent_info(self):
        return f"Tittel: {self._tittel}, forfatter: {self._forfatter}"
```

Instans-variabler

OOP – Eksempler objekter

```
def hoved():
    bok = Bok("Python for everyone", "Cay Horstmann", 10000)
    if bok.har_jeg_lest():
        print("Jeg har lest boken!")
    else:
        print("Jeg har ikke lest den")
    bok.les_bok()
    print("Nå har jeg lest boken:", bok.har_jeg_lest())
    print("Her er informasjon om boken:")
    informasjon = bok.hent_info()
    print(informasjon)

hoved()
```

OOP – Eksempler objekter

```
def hoved():
    bok = Bok("Python for everyone", "Cay Horstmann", 10000)
    if bok.har_jeg_lest():
        print("Jeg har lest boken!")
    else:
        print("Jeg har ikke lest den")
    bok.les_bok()
    print("Nå har jeg lest boken:", bok.har_jeg_lest())
    print("Her er informasjon om boken:")
    informasjon = bok.hent_info()
    print(informasjon)

hoved()
```

Kaller på konstruktøren

OOP – Eksempler objekter

```
def hoved():
    bok = Bok("Python for everyone", "Cay Horstmann", 10000)
    if bok.har_jeg_lest():
        print("Jeg har lest boken!")
    else:
        print("Jeg har ikke lest den")
    bok.les_bok()
    print("Nå har jeg lest boken:", bok.har_jeg_lest())
    print("Her er informasjon om boken:")
    informasjon = bok.hent_info()
    print(informasjon)

hoved()
```

bok er nå en variabel som har referanse til en objekt-instans

OOP – Eksempler objekter

```
def hoved():
    bok = Bok("Python for everyone", "Cay Horstmann", 10000)
    if bok.har_jeg_lest():
        print("Jeg har lest boken!")
    else:
        print("Jeg har ikke lest boken")
    bok.les_bok()
    print("Nå har jeg lest boken:", bok.har_jeg_lest())
    print("Her er informasjon om boken:")
    informasjon = bok.hent_info()
    print(informasjon)

hoved()
```

Vi bruker "dot-notasjon" for å koble oss til metoden i objektet

OOP – Eksempler objekter

```
def hoved():
    bok = Bok("Python for everyone", "Cay Horstmann", 10000)
    if bok.har_jeg_lest():
        print("Jeg har lest boken!")
    else:
        print("Jeg har ikke lest den")
    bok.les_bok()
    print("Nå har jeg lest boken:", bok.har_jeg_lest())
    print("Her er informasjon om boken:")
    informasjon = bok.hent_info()
    print(informasjon)

hoved()
```

Metodene fra grensesnittet

OOP – Eksempler klasser

```
class Konto:
    def __init__(self, navn, kontonummer):
        self._navn = navn
        self._kontonummer = kontonummer
        self._saldo = 0

    def sett_inn(self, beløp):
        self._saldo += beløp

    def ta_ut(self, beløp):
        self._saldo -= beløp

    def hent_saldo(self):
        return self._saldo

    def hent_navn(self):
        return self._navn
```

OOP – Eksempler objekter

```
konto = Konto("Miley Ray Cyrus", "05551234567")
assert konto.hent_saldo() == 0, "Konto må være 0 ved opprettelse"

konto.sett_inn(5000)
konto.ta_ut(799)
print(f"Kontoen til {konto.hent_navn()} har saldo: {konto.hent_saldo()}")

annen_konto = Konto("Brian Griffin", "05557238455")
print(f"Kontoen til {annen_konto.hent_navn()} har saldo: {annen_konto.hent_saldo()}")
```

OOP – Eksempler klasser

```
class Vare:
    def __init__(self, navn, pris):
        self._navn = navn
        self._pris = pris

    def pris(self):
        return self._pris

    def navn(self):
        return self._navn


class Handlekurv:
    def __init__(self):
        self._varer = []
        self._rabatt = 0

    def legg_til_vare(self, vare):
        self._varer.append(vare)

    def fjern_vare(self, vare_navn):
```

```
        vare = None
        for indeks in range(len(self._varer)):
            if self._varer[indeks].navn() == vare_navn:
                vare = self._varer[indeks]
        if vare:
            self._varer.remove(vare)

    def hent_innhold(self):
        ret = "Varer i kurven:\n"
        pris = 0
        for vare in self._varer:
            ret += f"{vare.navn():25} {vare.pris():5}\n"
            pris += vare.pris()
        if len(self._varer) == 0:
            ret += "Ingen varer i handlekurven\n"
        else:
           ret += f"Total {pris:25}\n"
        return ret
```

OOP – Eksempler objekter

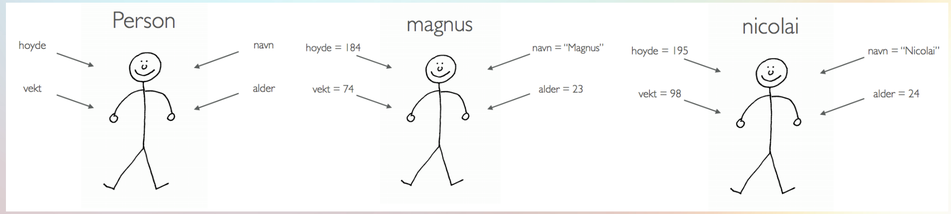
```
handlekurv = Handlekurv()

handlekurv.legg_til_vare(Vare("Ost", 25))
handlekurv.legg_til_vare(Vare("Kylling", 115))
print(handlekurv.hent_innhold())

handlekurv.fjern_vare("Ost")
print(handlekurv.hent_innhold())

handlekurv.fjern_vare("Kylling")
print(handlekurv.hent_innhold())
```

Oppgaver



Oppgave 1

Med utgangspunkt i tegningene lagd av personene ovenfor lag en klasse Person, med de gitte instansvariablene: navn, alder, høyde og vekt.

Oppgave 2

Vi skal nå utvide Person-klassen vår med følgende metoder:

1. `endre_navn(self, nytt_navn)` → endre navnet til nytt navn.
2. `skriv_ut_hilsen(self):` → skriv ut en hilsen fra personen, på formen “Hei, jeg heter <navn> og jeg er <alder> år gammel”.
3. `feire_bursdag(self)` →
   - Øk alderen til personen med 1.
   - Skriv ut en bursdagshilsen med den nye alderen.
   - Hvis personen fremdeles er under 18 år, skal også høyden økes med 2.

Oppgave 3

Gitt klassen Person som vi har laget over. I en egen fil:

1. Opprett 3 instanser (objekter) av klassen med navn/alder/vekt/høyde du velger selv.
2. Skriv deretter ut en hilsen fra hver person deres ved hjelp av skriv_ut_hilsen-metoden.
3. Feire bursdagen til én av personene, og skriv ut en ny hilsen fra denne personen.
4. Endre navnet til én av personene og skriv ut en ny hilsen fra denne personen.

Oppgave 4

Skriv en klasse Dyr. Et dyr har følgende egenskaper:

- Art
- Kjønn
- Vekt

1. Lag en konstruktør (init-metode).
2. Lag en metode som skriver ut informasjonen om dyret.
3. Opprett tre objekter av klassen Dyr i en annen fil, for å representere tre forskjellige dyr. Skriv ut informasjon om alle dyrene.