Uke 9, time 1

Onsdag 15. oktober

Introduksjon til objektorientert programmering

Dunder: spesielle metoder

`__eq__`

`__str__`

Spesielle metoder

- Python har en del forhåndsdefinerte metoder som vi kan implementere i klassene våre
- Disse kalles populært for dunder-metoder (hvor dunder er kortform for "double underscore") siden de starter og avsluttes med to understreks-tegn
- Blir ikke kalt på direkte, men brukes av Python gjennom ulike operatorer eller under spesielle tilfeller
- Allerede så har vi brukt `__init__` for å konstruere objektene
- Det finnes mange av de (over 100) men vi skal bare se på et par

`__eq__`

- Vi sammenligner verdier, og også objekter om de er **like** ved å bruke `==`
- Men hva betyr det?
- Vi har også sett på at vi må bruke `is` og `is not` når vi sjekker mot `None`
- `is` kalles for identitets-operator og sjekker om objektene på hver sin side er helt **identiske**, i praksis at de peker til samme referanse
- Men hvis vi vil sammenligne og vi synes f.eks. at sauer av samme farge bør regnes som like?
- Når vi skriver klasser så er det opptil oss å bestemme om to ulike objekter skal regnes som like
- Dette gjør vi med `__eq__` metoden

`__eq__`

- Det er metoden som blir kalt på om vi bruker == med instanser av klassen vår på en eller begge sidene
- `__eq__` har to parametere, den første er `self` og den andre viser til instansen som `self` skal sammenlignes med
- Metoden returnerer `True` hvis vi synes de er like, eller `False` hvis ikke
- Hvis vi ikke vet hvordan eller det ikke gir mening å sammenligne, så **må** vi returnere `NotImplemented`
- Når vi bruker == så vil Python først prøve `__eq__` på referansen til venstre. Hvis den svarer ikke implementert, så prøver den til høyre, og hvis ikke den heller ikke er implementert så vil den bruke identitets-operatoren `is` for sammenligning

__eq__

```
class Giraff:
  def __init__(self, farge):
    self._farge = farge

  def __eq__(self, annen):
    if not isinstance(annen, Giraff):
      return NotImplemented
    return self._farge == annen._farge

gul_giraff = Giraff("gul")
en_til_gul_giraff = Giraff("gul")
rosa_giraff = Giraff("rosa")

if gul_giraff == en_til_gul_giraff:                # True
  print("To like giraffer")

if gul_giraff == rosa_giraff:                      # False
  print("To like giraffer")

if gul_giraff == 2:                                # False
    print("Hva?")
```

`__eq__`

- Det er også mulig å lage en egen variant for `!=` ved å bruke `__ne__`, men hvis vi ikke implementerer det så vil resulatatet bare være `not __eq__`
- For de andre operatorene vi bruker for sammenligning, så har de også sine dunder-metoder:

`>`	`__gt__`
`<`	`__lt__`
`>=`	`__ge__`
`<=`	`__le__`

- Som et eksempel så kunne vi sammenlignet giraff-objekter etter hvor lang halsen er
- Ved å implementere disse, så kan en liste med objekter av klassen også sorteres ved å bruke `sort()`-metoden

`__str__`

- Om du bruker print direkte på en instans av en klasse, så får du typisk som svar noe som ligner på dette:

```
<__main__.Giraff object at 0x1011d1160>
```

- Det er ikke så ofte nyttig, så da kan vi bruke en metode som heller gir oss nyttig informasjon om objektet
- Den heter `__str__` og det eneste den skal gjøre er å returnere en string som skrives ut i stedet for det som er over

__str__

```
class Giraff:
  def __init__(self, farge):
    self._farge = farge

  def __str__(self):
    return f"Giraff som er {self._farge}"

gul_giraff = Giraff("gul")
print(gul_giraff)
```

`__str__`

- Det er en annen metode `__repr__` som også returnerer en string som gir informasjon om objektet
- Denne brukes for en mer teknisk representasjon
- Hvis vi ikke lager en __str__ så kalles denne, og har vi ikke en egen av denne, så er det en standard versjon som gir det fulle navnet til klassen og en id (typisk minne-adressen) til objekt-instansen
- Om vi lager egen variant, så er det forventet at det tilsvarer hvordan vi kan opprette et tilsvarende objekt programmatisk

__str__

```
class Giraff:
  def __init__(self, farge):
    self._farge = farge

  def __repr__(self):
    return f"Giraff('{self._farge}')"

gul_giraff = Giraff("gul")
print(repr(gul_giraff))
```

`__str__`

- Ingen av dunder-metodene er beregnet for å bli kalt på direkte
- For å hente ut stringen fra `__str__` så kan vi bruke `str(objektnavn)`
- Og tilsvarende for `__repr__` så finnes `repr(objektnavn)`
- Når vi bruker print, så er det `str(objektnavn)` som kjøres, og vi kan også bruke f-string: `f"{objektnavn!s} {objektnavn!r}"`

Spesielle metoder

- Som sagt så har vi veldig mange dunder-metoder, og de er ikke relevant for IN1000
- Men noen kan være nyttig å kjenne til, så vi har egne for de vanlige matematiske og logiske operatorene

`+`	`__add__`	`@`	`__matmul__`
`-`	`__sub__`	`&`	`__and__`
`*`	`__mul__`	`|`	`__or__`
`/`	`__truediv__`	`^`	`__xor__`
`%`	`__mod__`	`<<`	`__lshift__`
`**`	`__pow__`	`>>`	`__rshift__`
`//`	`__floordiv__`		

- Merk at de også har en variant som kalles om objektet er på høyre side av operatoren, som starter med bokstaven "r", så for `__add__` er tilsvarende `__radd__`, og hvis du har – eller + alene foran en verdi så er det hhv. `__neg__` og `__pos__`
- Og det er mange flere

Oppgaver

Oppgave 1

```
class Person:
    def __init__(self, fødselsnummer, navn):
        self._fødselsnummer = fødselsnummer
        self._navn = navn

    def __str__(self):
        return self._fødselsnummer + " (" + self._navn + ")"

print(Person("01010112345", "Indigo Matheus"))
```

a) Hva blir skrevet ut til terminalen ved kjøring av programmet over? Kom frem til et svar før dere kjører programmet.

b) Hva skjer hvis vi bruker print(self._fødselsnummer + " (" + self._navn + ")") i stedet for return self._fødselsnummer + " (" + self._navn + ")" ?

c) Hva blir skrevet ut til terminalen ved fjerning av __str__-metoden over? Kom frem til et svar før dere kjører programmet.

Oppgave 2

```
class Adresse:
    def __init__(self, gatenavn, bolignummer, postnummer):
        self._gatenavn = gatenavn
        self._bolignummer = bolignummer
        self._postnummer = postnummer

    def __eq__(self, annen):
        return (self._gatenavn == annen._gatenavn and
        self._bolignummer == annen._bolignummer and
        self._postnummer == annen._postnummer)

adr1 = Adresse("Pythongata", 1, 0000)
adr2 = Adresse("Pythongata", 1, 0000)
print(adr1 == adr2)
print(adr1 is adr2)
```

a) Hva skrives ut til terminalen med print(adr1 == adr2) i programmet over? Kom frem til et svar før dere kjører programmet.

b) Hva skrives ut til terminalen med print(adr1 is adr2) i programmet over? Kom frem til et svar før dere kjører programmet.

c) Sammenlign svarene i a) og b). Hva oppdager dere, og hvorfor er det slik?

d) Hvorfor trenger vi ikke egne metoder for å hente ut _gatenavn, _bolignummer og _postnummer fra annen-objektet i __eq__-metoden?

Oppgave 3

```
class Konsulentselskap:
    def __init__(self, organisasjonsnr, selskapsnavn, beskrivelse):
        self._organisasjonsr = organisasjonsnr
        self._selfskapsnavn = selskapsnavn
        self._beskrivelse = beskrivelse
```

a) Gitt klassen ovenfor, lag en __str__-metode med informasjon om konsulentselskap.

b) Lag en __eq__-metode som returnerer True hvis organisasjonsnr er likt, ellers False.

Oppgaver

Oppgave 4

```
class Matbutikk:
    def __init__(self):
        self._varer = []

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

    def hent_varer(self):
        return self._varer

class Vare:
    def __init__(self, vareid, varenavn):
        self._vareid = vareid
        self._varenavn = varenavn

def hovedprogram():
    skinkebutikk = Matbutikk()
    for i in range(5):
        ny_skinke = Vare(i, "Skinke-" + str(i))
        skinkebutikk.legg_til_vare(ny_skinke)

hovedprogram()
```

a) Utvid programmet ovenfor med en __str__-metode i Vare-klassen. Metoden skal inkludere både vareid og varenavn.

b) Fortsett å utvide programmet, slik at det ved hjelp av en annen for-løkke skriver ut vareid og varenavn til terminalen for varene i lista _varer refererer til. Dere skal dra nytte av __str__-metoden fra a) i løsningen (men aldri kalle på denne direkte).

c) Det er viktig at en vare med samme vareid og varenavn som en som allerede ligger i lista _varer refererer til, ikke kan legges til flere ganger. Utvid programmet med slik funksjonalitet ved å inkludere en __eq__-metode som del av løsningen. Hint: dere kan bruke str(self) og str(annen) i __eq__-metoden når to varer skal sammenlignes.

d) Hvorfor kunne dere ikke brukt str(self) og str(annen) i c) hvis Vare-klassen ikke hadde en __str__-metode?

Oppgaver

Oppgave 5

```
class Telefonnummer:
    def __init__(self):
        self._landskode = None
        self._nummer = None

    def sett_landskode(self, landskode):
        self._landskode = landskode

    def sett_nummer(self, nummer):
        self._nummer = nummer

    def __str__(self):
        return "+" + str(self._landskode) + " " + str(self._nummer)

def hovedprogram():
    telefonbok = {
                    "Iris Holter" : Telefonnummer(),
                    "Jørgen Olaisen" : Telefonnummer(),
                    "Laila Ringsaker" : Telefonnummer()
                }

hovedprogram()
```

a) Hva skjer hvis dere skriver ut Telefonnummer-objektet til Jørgen Olaisen? Forklar hvorfor det er slik.

b) Det skal kun være mulig å sette landskode hvis det ikke finnes fra før, og det skal kun være mulig å sette nummer hvis det ikke finnes fra før. Gjør nødvendige endringer i Telefonnummer-klassen for å møte disse kravene. Hvis landskode eller nummer (eller begge) finnes fra før, skal brukeren få beskjed om dette via utskrift i terminalen.

c) Utvid hovedprogrammet, slik at Iris Holter, Jørgen Olaisen og Laila Ringsaker alle får et telefonnummer bestående av landskode og nummer. Landskode skal være 47 (for Norge), men nummer velger dere selv.

d) Hovedprogrammet trenger også funksjonalitet for å skrive ut navn og telefonnummer for alle i telefonboka. Skriv kode for dette.

e) Prøv å sette landskode og nummer for Laila Ringsaker, selv om de finnes fra før.

f) Hvis en person i telefonboka skulle byttet telefonnummer (etter at de har fått både landskode og nummer), hvordan skulle man fått til dette?