Gruppetime

Uke 8

Agenda

- Mer objekt-orientering!
- Referanser
- None
- Datastrukturer
- Kontrollflyt

None

None brukes til å representere fraværet av en verdi. Altså hvis man har en variabel som referer til et objekt og ønsker at det ikke skal ha verdi kan man sette variablene til “None”.

Et eksempel på hvor None er brukt (dette er en del av et større program som skal programmeres senere, her har man en liste av student-objekter og ønsker å se om man finner studenten med det oppgitte navnet):

```
def finn_stud(self, navn):
        student = None
        for s in self._studentliste:
                if str(s) == navn:
                        student = s
        return student
```

Denne metoden vil enten returnere et objekt av typen student (dersom den finner riktig student), hvis den ikke finner den så vil den returnere None.

None

```
def finn_stud(self, navn):
    student = None
    for s in self._studentliste:
        if str(s) == navn:
            student = s
    return student
```

kan skrives om til



```
def finn_stud(self, navn):
    for s in self._studentliste:
        if str(s) == navn:
            return s
```

Hva er forskjellen mellom disse kodesnuttene?

None

None er også default-verdien som returneres fra enhver metode, med mindre noe annet er definert.

Med andre ord: lager vi en metode uten retuverdi, så ser det ut som den ikke returnerer noe, men i virkeligheten returnerer den “None”.

None

Fra der metoden er kalt på kan vi skrive:

```
student = <referansevariabel>.finn_stud(navn)
if student is None:
    print(“Fant ikke studenten”)
else:
    <gjøre andre ting, f.eks. kalle på metoder fra student-objektet>
```

Forsøker vi å kalle f.eks. **student.skriv_info()** og **student** er None vil programmet krasje.

Oppgave 1 - Oppvarmingsoppgave

Skriv en klasse sirkel. En sirkel har en radius; lag en konstruktør som setter radius.

Lag tre metoder til:

- en som returnerer diameteren til sirkelen
- en som returnerer omkretsen til sirkelen
- en som returnerer arealet til sirkelen.

*`Hint:` omkrets av en sirkel er diameter * pi, arealet av en sirkel er radius² * pi. Eksponenter skrives som <base>**<eksponent>, f.eks. 2^8 skrives 2 **8.*

Lag deretter 2 sirkler med ulik radius.
Skriv ut den ene sirkelens omkrets og areal, og den andre sirkelens diameter.

Oppgave 1 - løsningsforslag

```
import math (kun hvis man vil bruke math.pi)

class Sirkel:
      def __init__(self, radius) :
      self._radius = radius

      def diameter(self) :
      return self._radius*2

      def omkrets(self) :
      return self.diameter()* 3.14
            # eller (ikke pensum)
            return self.diameter()* math.pi

      def areal(self) :
      return (self._radius ** 2) * 3.14
```

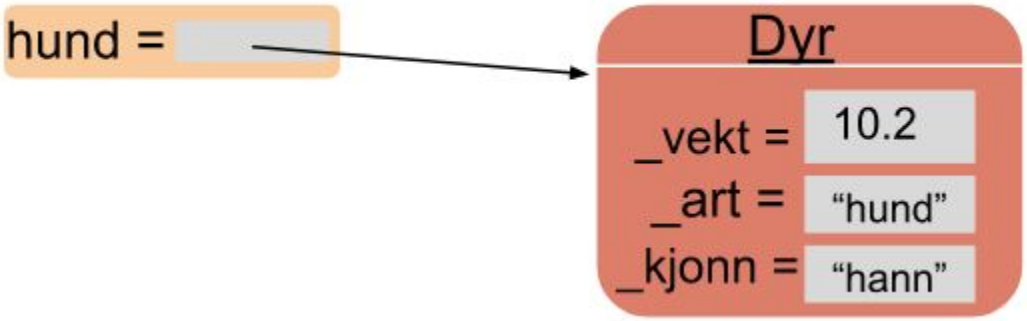
I hovedprogrammet:

```
from sirkel import Sirkel

sirkel1 = Sirkel(2)
sirkel2 = Sirkel(4.5)
print("sirkel1, omkrets:", sirkel1.omkrets(),"areal:", sirkel1.areal())
print("sirkel2, diameter:",sirkel2.diameter())
```

Datastruktur og kontrollflyt

```
hund = Dyr("hund", "hann", 10.2)
katt = Dyr("katt", "hunn", 5.7)
sau = Dyr("sau", "hann", 30.6)
```



Hva skjer hvis vi skriver `katt = hund` ?

Datastruktur og kontrollflyt

Fra forelesning:

Kode og tegninger

- variabler tegnes som en navngitt boks (rektangel) med verdi inni
- hvis verdien er en int, boolean eller string skriver vi den vanligvis direkte inne i boksen (selv om string er et objekt vi kan kalle metoder på)



- objekter (av mutable/ forandelige klasser) tegnes som rektangler med runde kanter
- ofte angir vi klassen (typen) over objektet – da med : foran klassenavnet
- referanser tegnes som piler fra referansevariabel til objekt
- instansvariabler tegnes inne i objektet: Som bokser eller bare <_instansvariabel> = <verdi>



Referanser

Vi bruker referanser for å få tak i objekter.

- En referanse er noe (f.eks. et navn) som peker på/henviser til en verdi eller et objekt i minnet.

For eksempel

hund = Dyr(“hund”, 23, 2.5)

Her er “hund” en referanse til et objekt av typen dyr.

“hund” *ER* ikke selve dyreobjektet, *men* ***refererer*** *kun til det*.

Self

Self brukes til å referere til den instansen av klassen metodene ble kalt på (eller for konstruktøren: det nye objektet som blir opprettet).

Instansvariablene defineres som self._navn for at de skal være instansvariabler og ikke vanlige lokale variabler.

Lokale variabler (og parametere) kan bare brukes i metoden/funksjonen de er definert i - mens instansvariablene kan brukes av alle metodene i en klasse (fordi de har self).

Oppgave 2

Lag en klasse Blomst. En blomst har en **art,** en **høyde**, en **instansvariabel som sier om blomsten har det bra eller ikke** (denne verdien er en **boolean**), og en **verdi som forteller hvor lenge siden blomsten ble vannet sist**. Konstruktøren setter alle disse verdiene.

- Når en blomst blir opprettet vil blomsten alltid ha det bra og det vil være 0 dager siden den ble vannet sist.

En blomst har en metode hent_status som returnerer verdien som forteller om blomsten har det bra eller ikke.

I tillegg har den metoden skriv_ut_info, som skriver ut en info streng om blomsten.

Videre har en blomst en metode neste_dag, som øker antall dager siden den ble vannet med en. Hvis det er mer enn tre dager siden blomsten ble vannet sist vil statusen til blomsten være at den har det dårlig. Hvis statusen til blomsten er at den har det bra vokser blomsten 1 cm.

Den siste metoden en blomst har er metoden vann. Hvis det er mindre enn 3 dager siden blomsten ble vannet vil statusen til blomsten bli dårlig (over vanning), ellers vil statusen være bra.

Skriv denne klassen + lag et testprogram for å teste klassen din

Minnegaten!