Skolan för Datavetenskap och kommunikation

DD1319
Programmeringsteknik

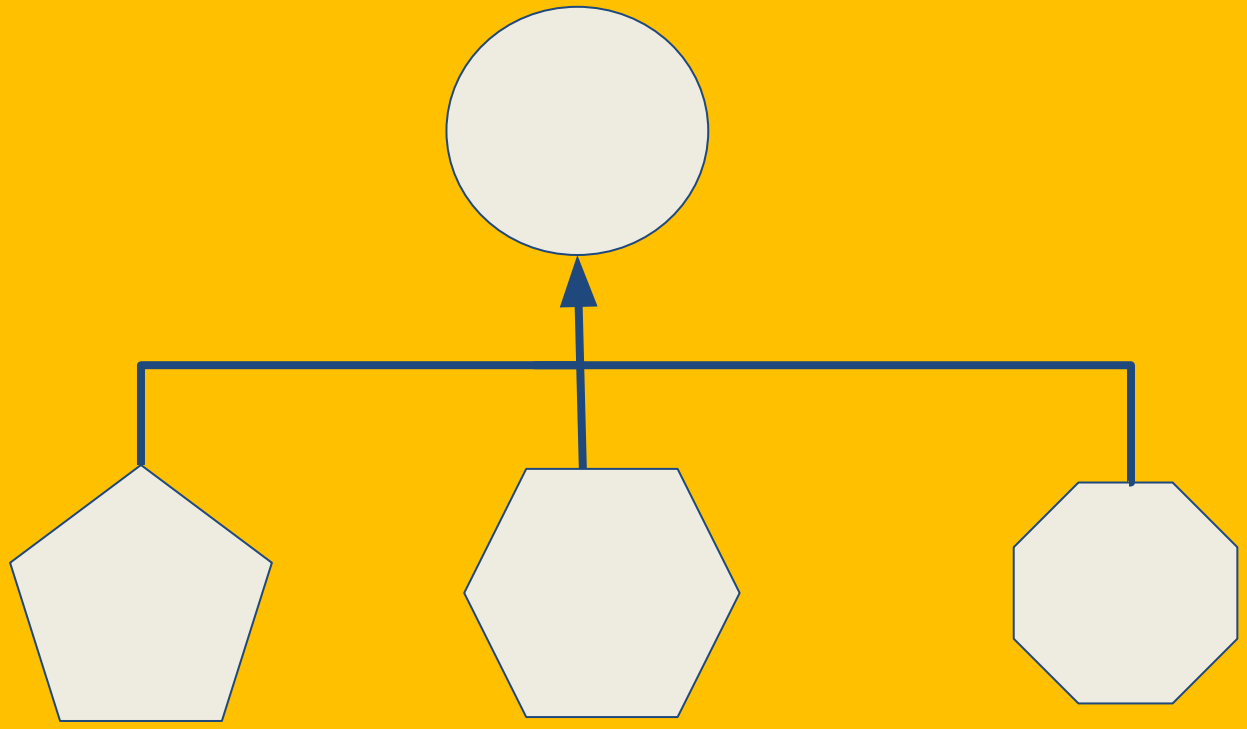
Föreläsning 10

idag

- Arv
- Överlagring
- super()
- Klassattribut
- Set

Arbetsblad föreläsning 10

arv



arv

```
class Subklass(Superklass):
```

Subklassen ärver alla

- attribut
- metoder

från klassen Superklass.

Klassen Bok från förra föreläsningen

```
class Bok:

    def __init__(self, titel, författare):
        self.titel = titel
        self.författare = författare

    def __str__(self):
        return "'"+self.titel + "' skriven av " + self.författare


def main():
    bok1 = Bok("Blackout", "C. Willis")
    print(bok1)


main()
```

Uppgift: Lista av Bok-objekt

1. Skapa tre Bok-objekt
2. Lägg in de tre bok-objekten i en lista
3. Skriv ut listan

utskrift av lista med objekt

`print(boklista)` ger utskriften

```
[<Bok object at 0x03D06550>, <Bok object at
0x04163DF0>, <Bok object at 0x04163FD0>]
```

problemet

Vi vill att `print(boklista)` ska skriva ut böckerna, t ex

```
'Blackout' skriven av C. Willis
'Piranesi' skriven av S. Clarke
'Head On' skriven av J. Scalzi
```

lösning

Vi vill ha en lista som för varje objekt anropar objektets `__str__` vid utskrift.

Definiera en ny klass `Boklista` som har en `__str__` som fungerar så.

Den kan vara en subklass till `list`.

överlagring

Vi kan definiera om en metod i en subklass.

Det är alltid den närmaste definitionen (dvs subklassens definition, inte superklassens) som används.

Boklista (ärver från list)

```
class Boklista(list):

    def __str__(self):
        s = ""
        for bok in self:
            s += str(bok) + "\n"
        return s
```

Uppgift: vad är self här?

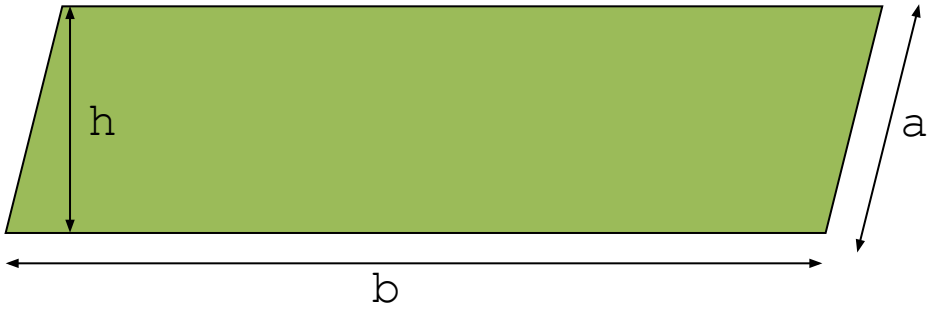
```
class Boklista(list):

    def __str__(self):
        s = ""
        for bok in self:
            s += str(bok) + "\n"
        return s
```

Provkör

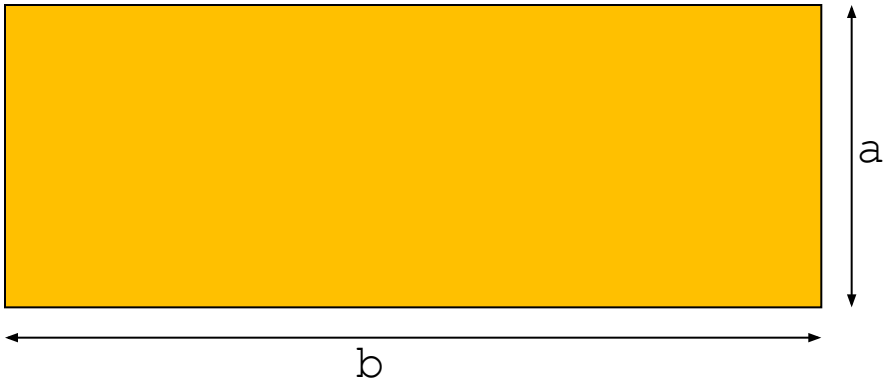
```
def main():
    lista = [Bok("ABC","Elmer"),\
    Bok("Varvet","N Båth"),\
    Bok("Matte","L Ullemar")]

    print("Utskrift av 'list'")
    print(lista)

    boklista = Boklista(lista)
    print("Utskrift av 'Boklista'")
    print(boklista)
```

geometri-exempel

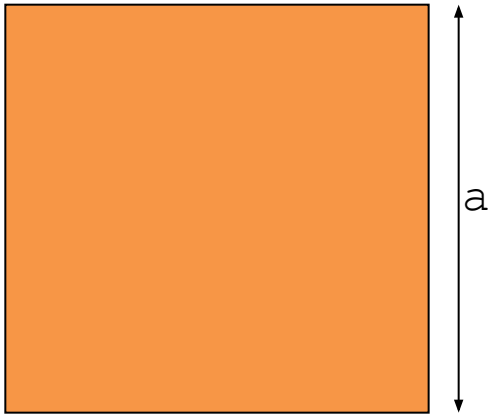
- *Parallellogram* är den mest generella figuren - den får bli superklass
- *Rektangel* är en sorts *parallellogram* med räta vinklar - vi låter den vara subklass till Parallellogram
- *Kvadrat* är en sorts *rektangel* med lika sidor - den får vara subklass till Rektangel



Parallellogram:
area = `h*b`
omkrets = `2*(a+b)`



Rektangel:
area = `a*b`
omkrets = `2*(a+b)`



Kvadrat:
area = `a*a`
omkrets = `4*a`

```
class Parallellogram(object):

    def __init__(self,a,b,h):
        self.kant1 = a
        self.basKant = b
        self.hojd = h

    def area(self):
        return self.hojd*self.basKant

    def omkrets(self):
        return 2*(self.kant1+self.basKant)
```

```
class Rektangel(Parallellogram):

    def __init__(self,a,b):
        self.kant1  =  a
        self.basKant = b

    def area(self):
        return self.kant1*self.basKant
```

```
class Kvadrat(Rektangel):

    def __init__(self,a):
        self.kant1 = a
        self.basKant = a
```

Kvadrat-objekt

Vi skapar en instans av Kvadrat...

... och anropar `area()` och `omkrets()`.

Vilka metoder (i vilken klass) är det som används?

```
# Huvudprogrammet

from geometri import *

def main():
    sida = input("Hur stor är sidan? ")
    kvadrat = Kvadrat(sida)
    print("Arean = ", kvadrat.area())
    print("Omkretsen = ", kvadrat.omkrets())

main()
```

ärva allt

Det går bra att skapa en egen klass som ärver allt från superklassen.

```
class Sjutton(Exception):
    pass
```

pass betyder "tom sats" - inget händer här.

```
class Sjutton(Exception):
    pass

def skriv_tal(antal):
    for i in range(antal):
        if i == 17:
            raise Sjutton
        else:
            print(i)

def main():
    try:
        skriv_tal(100)
    except Sjutton:
        print("Det var som sjutton!")

main()
```

super

Med hjälp av `super()` går det att komma åt superklassens metoder.

```
class Papper:

    def __init__(self):
        self.färg = "vitt"
        self.format = "A4"
        self.ytvikt = 80
        self.hålat = False


class Låda(Papper):

    def __init__(self):
        super().__init__()
        self.antal_paket = 5
```

Vad händer om Pappers init har parametrar?

```
class Papper:

    def __init__(self,a,b,c):
        self.färg = "vitt"
        self.format = "A4"
        self.ytvikt = 80
        self.hålat = False


class Låda(Papper):

    def __init__(self,a,b,c):
        super().__init__(a,b,c)
        self.antal_paket = 5
```

klassattribut

- Ett klassattribut är en variabel som är gemensam för alla objekt i en klass.
- Om värdet ändras för ett objekt ändras det för alla.
- Klassattribut går också att komma åt oberoende av objekten.

```
class Ägg:

    antal = 0   # antal är ett klass-attribut

    def __init__(self):
        Ägg.antal += 1
        print("Detta är ägg nr ", Ägg.antal)


def main():
   for i in range(7):
      x = Ägg()

main()
```

```
class Långord:

    VOKALER = "aeiouyåääö"  # VOKALER är ett klassattribut

    def __init__(self, ordet):
        self.ordet = ordet

    def antal_vokaler(self):
        antal = 0
        for bokstav in self.ordet:
            if bokstav in Långord.VOKALER:
                antal += 1
        return antal

def main():
   ordobjekt = Långord("ortnamnsforskning")
   print(ordobjekt.antal_vokaler())

main()
```

set

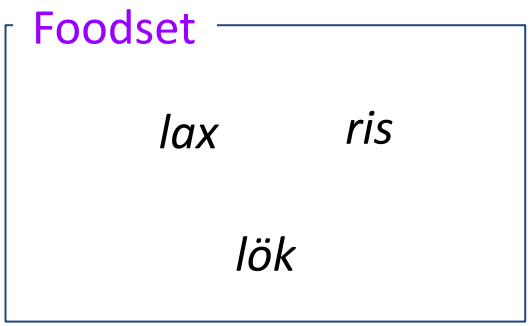
I labb 6 ska ni använda en ny datatyp: *set*

Träna på att läsa

dokumentationen för set

labb 6

Uppgift 1



labb 6

Uppgift 2

Pannkakor

Foodset

gluten *mjölk* *ägg*

smör *vetemjöl*

salt

labb 6

Uppgift 3

labb 6

Uppgift 4