# GruProg DD1331

## Övning 3

Anna Karlhede
karlhed@kth.se

Repo: https://github.com/TypAnna/GruProgDD1331

# Idag

- Repetition/genomgång
  - Objekt
  - Klasser
- Uppgifter
  - rekursion

# Repetition

# Klasser och objekt

Objekt är en instans av (någon) datastruktur.
- på detta objekt kan vi utföra diverse metoder
  - tex: en lista är ett objekt, och på listor kan vi använda metoden `append()` för att lägga till element, `[1,3,].append(5)`

Hur dessa objekt ska bete sig (tex vilka metoder vilka metoder vi kan anropa på dom, och andra egenskaper) bestäms av objektets klass. Tänk på klasser som en mall!

Några inbyggda klasser: int, str, list, float
… och några instanser (objekt) av dessa klasser: `3` (int), `"hejhej"` (str), `["katt", "hund"]` (list), `2.34` (float)

Du kan skapa helt nya sorters objekt, som beter sig på ett sätt du själv bestämmer, genom att göra nya klasser!

# Att skapa egna klasser och objekt

Låt säga att vi ska göra ett program där en styr en bil. Vi vill att en bil ska ha en färg, ett märke, en maxhastighet, och att hastigheten ska kunna ändras.

För detta kan vi skapa en ny klass - en bilklass!

se filen `carClass.py` (och `carClassDoc.py` för bra dokumentation)

En klass är en mall, en ritning, för hur ett objekt ska bete sig. Från denna mall kan vi skapa massa olika objekt!

# Klass, objekt, typ - vad är vad?!

Klass: en mall (blueprint) från vilken objekt kan skapas.

Objekt: en instans av en klass

Car är en **klass**

myCar är ett **objekt** (pekar mot ett objekt ifall vi ska vara helt korrekta) är en instans av klassen Car

myCar har **typen** Car

För att läsa mer, se här.

```
class Car:
    #konstruktor
    #andra metoder

myCar = Car()
```

# funktion vs metod i pyton

En funktion är ett kodblock som utför någon uppgift.

En metod är en funktion associerad med ett objekt/en klass.

`changeSpeed()` i vår `Car`-klass är alltså en metod.

# Rekursion

Att kalla en funktion, inuti funktionen självt, kallas rekursion.

Rekursiva metoder kan vara väldigt kraftfulla, och blir ofta snygga (lättlästa).

Rekursiv programmering och matematisk induktion hänger ihop. Vi har ett (eller flera) basfall, och en term bestäms på något vis av de tidigare termerna. Tänk på tex n! eller fibonaccis talföljd.

En uppgift som tex fibonaccis talföljd kan lösas både med och utan rekursion. Vilken är bäst?

# Uppgifter

# Uppgift 1- Fibonaccis talföljd

Beräkna det n:te fibonacci-talet rekursivt.

# Uppgift 2 - Rita trianglar

Skriv en funktion triangel_bas_upp(dist, bredd) som ritar en triangel med asterisker.
Anropet triangel_bas_ner(1, 7) ska ge

```
 *******
  *****
   ***
    *
```

Skriv också  triangel_bas_ner(dist, bredd)

# Uppgift 3 - Beräkna siffersumman

Skriv en metod som beräknar siffersumman av ett heltal rekursivt.

# Uppgift 4 - Är listan stigande?

Med hjälp av rekursion, tag reda på om elementen i en lista är icke-avtagande, dvs sorterade i stigande ordning. T.ex. är båda listorna [1,1,1,1] och [-4,-3,-2,0,0,1] icke-avtagande.

# Uppgift 5 - Potensräkning

Beskriv en algoritm för att beräkna x^n rekursivt m.h.a. multiplikationer. n är ett heltal.

# Uppgift 6 - Permutationer

Konstruera alla permutationer av en lista eller textsträng. T.ex. om man startar med listan [1,2,3] så är permutationerna [[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]].

# Uppgift 7 - repetition

Vad kommer detta program skriva ut?

```
s = 'Hello World'
s.upper()
s[6:11]
print(s)
```